Titre: Title:	Analyse et conception de registres à décalage pour la réalisation de décodeurs à seuil itératifs configurables
Auteur: Author:	Martin Dubois
Date:	2004
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Dubois, M. (2004). Analyse et conception de registres à décalage pour la réalisation de décodeurs à seuil itératifs configurables [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7369/

UNIVERSITÉ DE MONTRÉAL

ANALYSE ET CONCEPTION DE REGISTRES À DÉCALAGE POUR LA RÉALISATION DE DÉCODEURS À SEUIL ITÉRATIFS CONFIGURABLES

MARTIN DUBOIS

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE EN SCIENCES APPLIQUÉES

(GÉNIE ÉLECTRIQUE)

AOÛT 2004

Your file Votre référence
ISBN: 0-494-01312-5
Our file Notre référence
ISBN: 0-494-01312-5

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE

Le titre du mémoire :

ANALYSE ET CONCEPTION DE REGISTRES À DÉCALAGE POUR LA RÉALISATION DE DÉCODEURS À SEUIL ITÉRATIFS CONFIGURABLES

Présenté par : Martin Dubois

En vue de l'obtention du diplôme de : Maîtrise en sciences appliquées

A été dûment accepté par le jury de l'examen constitué de :

M. Christian Cardinal, Ph.D., Président

M. Yvon Savaria, Ph.D., membre et directeur de recherche

M. David Haccoun, Ph.D., membre et codirecteur de recherche

M. Mohamad Sawan, Ph.D., membre

REMERCIEMENTS

Je remercie mon directeur de recherche, Yvon Savaria, pour son encadrement et ses suggestions lors de la réalisation de ce projet, ainsi que mon codirecteur de recherche David Haccoun pour son encadrement, ses conseils et son soutien financier. De plus, je remercie Christian Cardinal pour ses conseils et ses suggestions pendant l'élaboration de ce projet de recherche et de sa réalisation.

Je tiens à remercier tous les autres membres de l'équipe tel que Marc-André Cantin, Professeur Mohamad Sawan, Ghislain Provost et Alexandre Raymond pour leurs collaboration et contributions au projet. Plus particulièrement, j'aimerais remercier Normand Bélanger pour son aide et ses précieux conseils pour la rédaction des articles. Mes remerciements vont aussi au personnel administratif tel que Ghyslaine E-Carrier et Claudine Ouellet.

Enfin, je remercie mes parents et mon frère Mathieu Dubois pour leur soutien dans ce projet de recherche.

RÉSUMÉ

Le décodage à seuil itératif est une approche simple permettant de contourner les problèmes de latence et de complexité du décodage Turbo. Ce projet de maîtrise consiste à concevoir un décodeur à seuil itératif configurable. Un décodeur configurable permet d'avoir plusieurs décodeurs dans un design en ajustant le nombre de connexions. Cet ajustement permet de changer de performance dans un contexte donné sans avoir à synthétiser un nouveau design. Évidemment, l'adaptation de ce décodeur configurable est d'un grand attrait parce qu'il permet d'obtenir des meilleures performances en terme de probabilité d'erreur. Ce dernier facteur est influencé par le rapport signal sur bruit du canal qui est parfois variable pour certaines applications.

Comme le décodeur est composé d'une architecture pipeline, le travail a été séparé en deux grandes parties. Ceci a permis d'écrire deux articles qui correspondent respectivement au chapitre 3 et au chapitre 4. La première partie discutera de nouvelles méthodes et structures pour la conception et la réalisation de registres à décalage de petite, moyenne et grande taille. Cette partie se concentrera sur l'aspect de la réduction de puissance des registres à décalages et la possibilité d'avoir des structures consommant le moins d'énergie possible tout en étant configurables. Avec ces nouvelles méthodes et structures, la réalisation d'un décodeur configurable deviendra possible. La deuxième partie discutera de la façon de concevoir un décodeur configurable. Les méthodes actuelles ne sont pas adéquates pour cette conception. Cette partie se concentrera sur les méthodes et sur la modélisation de plusieurs décodeurs afin de les fusionner. De plus, il

existe plus d'un code pour chaque décodeur, le concept d'exploration de la fusion afin de trouver le meilleur ensemble de codes pour une performance donnée sera montré.

ABSTRACT

Iterative threshold decoding is an easy approach to circumvent the latency problems and complexity of Turbo decoding. This thesis explores the design of a class of configurable iterative threshold decoders. A configurable decoder allows having multiple decoders in a system at a cost smaller than the sum of the individual decoders it can implement. It allows changing the code and the error performance without having to synthesize a new design. Obviously, the configurable decoder is very attractive because it can be adjusted to give the needed performance in term of error probability, while minimizing power consumption when less powerful codes are sufficient. Selection of the optimal decoder is influenced by the channel signal to noise ratio. The decoder is a heavily pipelined architecture. This work is split in two main parts that led to two articles presented in chapter 3 and chapter 4. The first part discusses new methods and structures to design and implement small, medium size and long shift registers. This part will focus on power reduction. It will also propose means to get configurable size structures. With these new methods and structures, configurable decoders can be easily implemented. The second part will propose systematic methods to generate configurable decoders by merging together several decoders. The existing design methods are not adequate for this task. As several codes exist for the same decoder size, this allows exploring the performance of various codes more effectively.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES NOTATIONS ET SYMBOLES

ASAP : As soon as possible

ALAP : As late as possible

ALU : Arithmetic logic unit

AR : Architecture reconfigurable

ASIC : Application-Specific Integrated Circuit

CSO2C : Convolutional self-doubly orthogonal codes

DSP : Digital signal processing

LFSR : Linear feedback shift register

FDS : Force-directed scheduling

RAM : Random Access Memory

RTL : Register transfer level

SRL16 : Left shift register 16 bit

VHDL : Very high speed integrated circuit Hardware Description Language

VLSI : Very large scale integration

AVANT-PROPOS

Cette recherche a été effectuée en étroite collaboration entre deux groupes de recherche de l'École Polytechnique de Montréal. Le premier groupe s'intéresse à la recherche du décodeur et du codeur permettant d'avoir des meilleures performances sur un canal de transmission. Le deuxième s'intéresse aux méthodes et aux implémentations d'un ou plusieurs décodeurs.

Cette recherche vise deux principaux objectifs soit la création de registres à décalages configurables tout en minimisant la consommation de puissance et des méthodes pour concevoir un décodeur configurable. Ces méthodes permettent de modéliser plusieurs décodeurs tout en permettant d'implémenter un décodeur configurable.

CHAPITRE 1

INTRODUCTION

1.1 Motivations

Le domaine des télécommunications demande de plus en plus des systèmes performants pour la correction d'erreurs. Ils ont besoin de latence faible et de taux de transfert élevé. Il y a aussi une tendance à adapter les systèmes aux canaux de communication. Par conséquent, une technique efficace et pratique de correction d'erreurs utilisant les codes convolutionnels, basée sur des codes doublement orthogonaux, traités avec le décodage à seuil itératif a été proposée par Cardinal, Haccoun et Gagnon [6], [7]. Cette approche simple permet de contourner les problèmes de latence et de complexité du décodage Turbo conventionnel [6]. Donc, le premier objectif est de connaître la meilleure approche pour réaliser ce type de décodeur. Comme ce décodeur comporte principalement des registres à décalage, compte tenu de la grande activité que ces registres peuvent générer, il est nécessaire d'avoir des structures qui consomment le moins possible de puissance. Par la suite, il faut les rendre configurables pour leur intégration dans un décodeur configurable, ce qui représente notre second objectif. Le décodeur peut être utilisé dans deux principales applications. La première application est pour la recherche. À titre d'exemple, un décodeur configurable peut valider cinq codes au lieu de nécessiter la mise en œuvre de cinq décodeurs. Cette opération permet de ne pas passer par cinq processus de conception et de synthèse. La

deuxième application est de prendre un ensemble de codes pour en faire un décodeur configurable qui peut s'adapter à son environnement ou à son canal de communication.

1.2 Contributions

Les principales contributions de ce projet sont :

- La création de méthodes et de structures pour la réalisation de registres à décalage qui consomment moins de puissance tout en étant configurables.
- La création de méthodes pour la conception de décodeurs configurables et d'un outil qui implémente ces méthodes.

1.3 Organisation du mémoire

Le chapitre 1 présente une revue de la littérature sur les registres à décalage statiques et configurables. De plus, le décodeur, les méthodes de conception traditionnelles et les architectures configurables seront revus brièvement. Le chapitre 2 couvre l'aspect de la réduction de la consommation de puissance des registres et propose des méthodes pour les rendre configurables. Deux nouvelles structures sont présentées. Ces structures configurables permettent une réduction de puissance importante. Par conséquent, nous proposons une méthode pour choisir la meilleure structure pour une configuration donnée de registres à décalage. Ceci sera fait par un article que nous présentons. Le chapitre 3 consiste à l'introduction des méthodes pour l'exploration et la conception d'un décodeur configurable sous forme d'article. La première phase étant de modéliser un décodeur, la deuxième phase montre des méthodes pour fusionner plusieurs décodeurs ensemble. La dernière étape est de choisir le meilleur ensemble de codes pour

une performance donnée, ce qui nous amène à l'exploration des décodeurs fusionnés ou non.

Enfin, le chapitre 5 présente nos conclusions. Nous résumons l'ensemble des travaux réalisés et nous suggérons des pistes de travaux futurs pour cette recherche.

CHAPITRE 2

REVUE DE LA LITTÉRATURE

2.1 Principes de codage et décodage

Dans cette section, nous décrivons les concepts de codage et de décodage. Notre intention étant de donner un aperçu sommaire afin de comprendre les bases et non pas de faire une étude approfondie sur les façons de faire. En première partie, nous introduisons la notion de codage de canal dans un système de communication numérique. Ensuite, nous décrivons la façon de décoder pour finalement se concentrer sur la version du décodeur qui nous a servi de référence.

2.1.1 Codage

Dans cette section, nous faisons un survol d'un modèle de communication numérique et de la façon avec laquelle on code l'information afin de la récupérer à la sortie du système de communication. Tout d'abord, la Figure 2.1 illustre un modèle simplifié d'un système de communication numérique. I, C, C' et I' représentent respectivement l'information à transmettre, l'information encodée, l'information reçue du canal et l'information récupérée.

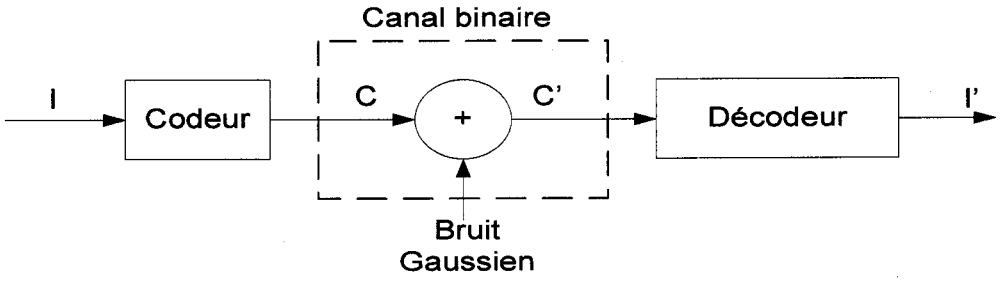

Figure 2.1 Modèle simplifié d'un système de communication numérique

Dans tout système de communication, il y a du bruit. Ce bruit affecte l'information transmise dans le canal. Selon le type de canal, nous choisissons le type de codage et de décodage. L'objectif ultime étant d'avoir I=I'. Par conséquent, l'objectif du codage et du décodage est de réussir à contrôler les erreurs qui sont introduites par le bruit sur le canal. Bien sûr, ce n'est pas le seul facteur pour créer des erreurs sur l'information transmise. Pour nos besoins, nous conservons cette simplicité. Par conséquent, un système de communication avec codage peut être évalué par sa capacité de corriger les erreurs, son taux de transmission et sa complexité. Le processus de codage est relativement simple par rapport au processus de décodage. Le codage est le processus qui produit à chaque instant des symboles qui sont transmis dans le canal de communication. Par exemple, nous transmettons sur le canal l'information et la parité tel qu'illustré sur la Figure 2.2. Ceci est pour un codage avec un code particulier appelé convolutionnel doublement orthogonal et dénoté CSO2C. Doublement orthogonal signifie que les différences entre toutes les différences sont distinctes.

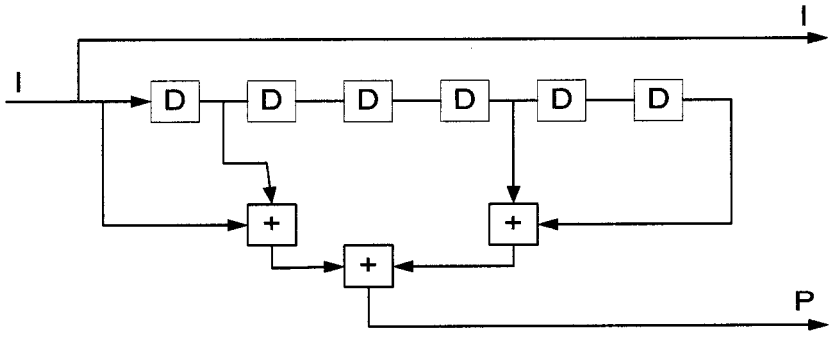

Figure 2.2 Exemple de codeur CSO2C J=4

I, P, + et D sur la Figure 2.2 représentent respectivement le symbole d'information, le symbole de parité, un opérateur modulo-2 et une bascule. J représente le nombre d'équations de parité. Pour chaque J, il existe plusieurs CSO2C possible, mais en général, on choisit celui de longueur minimale. Dans notre exemple, J=4 et un ensemble constitué de 0, 1, 4, 6 définit les interconnexions au codeur aussi bien qu'au décodeur.

L'algorithme de codage est relativement simple, il suffit de faire un modulo-2 entre certains signaux extraits du registre à décalage pour produire le symbole P. Ainsi, avec l'information et la parité, un décodeur peut grâce à un algorithme de décodage corriger les erreurs et récupérer I, l'information originale.

2.1.2 Décodage

L'algorithme de décodage permet de corriger les erreurs et de récupérer l'information originale. Cependant, cette partie est la plus complexe. Les méthodes actuelles telles le décodage de Viterbi et le décodage Turbo sont complexes et ont une latence importance, en particulier le décodage Turbo. La latence est définie par le temps qu'il faut au système pour sortir l'information par rapport à son entrée. Pour certaines applications en temps réel, cela peut être critique. Par exemple, lorsqu'on parle au téléphone, on ne veut pas avoir trois à quatre secondes d'intervalle entre chaque interlocuteur. D'autre part, pour des systèmes qui reçoivent des images de l'espace la latence est moins importante. On peut alors se permettre des systèmes très complexes pour récupérer les informations bruitées. Par conséquent, une technique efficace et pratique de correction d'erreurs utilisant les codes convolutionnels doublement orthogonaux, avec décodage à seuil

itératif a été proposée par Cardinal, Haccoun et Gagnon [6], [7]. Cette approche simple permet de contourner les problèmes de latence et de complexité du décodage Turbo conventionnel. C'est un décodage symbole par symbole. Cette méthode se distingue des autres parce qu'elle ne nécessite aucun entrelacement ni au codage et ni au décodage. De plus, cette méthode itérative est plus simple et elle permet une réduction de la latence. Le décodeur peut avoir un ou plusieurs étages. Ces étages sont connectés en cascade et selon nos besoins, nous choisissons le nombre d'étages requis. Le premier étage est différent des autres en termes d'équations. La Figure 2.3 montre le premier étage d'un décodeur à 8 étages pour J=4.

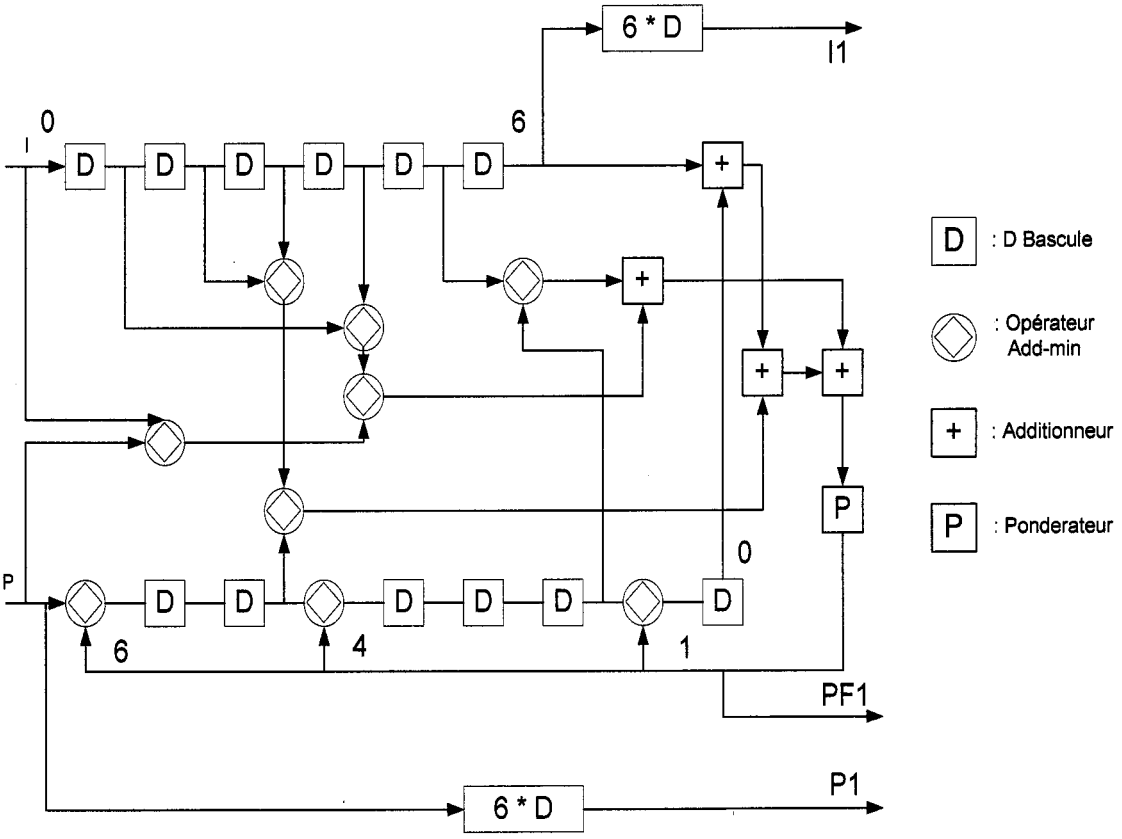

Figure 2.3 Décodeur à seuil avec J=4

Dans la Figure 2.3, I, P, D, +, et le losange dans un cercle représentent respectivement l'information, la parité, une bascule , un additionneur et un opérateur add-min. L'opérateur add-min représenté avec un losange est défini comme étant :

$$A \Diamond B = -sign(A)sign(B)Min(|A|,|B|) \tag{2.1}$$

Comme nous pouvons le remarquer, le décodeur comporte aussi un codeur. C'est-à-dire qu'une copie du codeur est en sorte imbriquée dans le décodeur. Le décodeur est construit selon un code particulier et un J donné . Le Tableau 2.1 liste des codes que nous utilisons dans le cadre de notre recherche [6], [7] et [16].

Tableau 2.1 Codes convolutionnels doublement orthogonaux tirés de [6], [7] et [16]

J	Codes	J	Codes
2	0,1	7	0,1,53,128,207,216,222 0,1,9,71,177,215,222 0,13,51,157,219,227,228 0,1,13,83,196,201,229 0,28,33,146,216,228,229 0,1,14,74,178,227,231 0,4,53,157,217,230,231 0,1,5,141,193,216,232 0,16,39,91,227,231,232 0,12,54,71,200,223,232 0,9,32,161,178,220,232
3	0,2,5	8	0,43,139,322,422,430,441,459 0,18,29,37,137,320,416,459 0,13,102,146,393,454,459,461 0,2,7,68,315,359,448,461 0,22,73,87,209,427,473,474 0,1,47,265,387,401,452,474 0,9,44,82,110,342,451,475 0,24,133,365,393,431,466,475
4	0,1,4,6 0,4,14,15	9	0,9,21,395,584,767,871,899,912 0,13,41,145,328,517,891,903,912 0,2,140,267,304,822,858,916,943 0,27,85,121,639,676,803,941,943 0,34,48,60,386,743,836,935,979 0,4,155,191,216,330,857,969,999 0,30,144,669,783,808,844,995,999 0,101,118,269,459,944,1021,1025,1044 0,19,23,100,585,775,926,943,1044 0,7,23,241,419,740,980,1034,1046 0,12,66,306,627,805,1023,1039,1046
5	0,1,24,37,41 0,4,17,40,41	10	0,29,40,43,1020,1328,1495,1606,1696,1698 0,27,93,503,600,1247,1646,1714,1825,1835 0,52,144,186,564,667,1120,1748,1814,1835 0,49,115,743,1196,1299,1677,1719,1811,1863 0,92,104,176,339,1311,1325,1738,1845,1876 0,31,138,551,565,1537,1700,1772,1784,1876 0,176,200,338,557,1375,1401,1844,1871,1876 0,5,32,475,501,1299,1538,1676,1700,1876
6	0,51,64,76,78,98 0,1,17,70,95,100 0,5,30,83,99,100 0,2,13,72,97,102 0,5,30,89,100,102	11	0,220,521,695,908,926,1059,2457, 3367,3458,3490
12	0,48,212,1014,1381,2217,4198,4373,4766,4885,4914,5173		

Notre objectif n'est pas de construire de nouveaux codes, mais plutôt de les implémenter en pratique. Par exemple, si nous voulons implémenter le code J=4, nous prenons J=4 avec le code 0, 1, 4, 6 tel qu'illustré sur la Figure 2.3 pour le premier étage. Pour ce qui est des interconnexions du registre de parité. Nous partons de la droite vers la gauche, et chaque position donne un add-min incorporé dans le registre à décalage et selon le cas, nous construisons les interconnexions pour les connecter aux add-min. Pour ce qui est du registre d'information nous obtenons la structure dont les caractéristiques sont résumées au Tableau 2.2.

Tableau 2.2 Construction d'un registre d'information

Différence	Position, # ports sur add-min	Différence	Position, # ports sur add-min
6 - 4	2, 4	4 – 1	3, 3
6 - 1	5, 4	4 – 0	4, 3
6 - 0	6, 4	1 – 0	1, 2

Il existe plusieurs codes pour un J donné. La performance du décodeur va varier selon J et le nombre d'itérations et non selon le code. Par exemple, les codes 0, 1, 4, 6 et 0, 4, 14, 15 pour J=4 donnent la même performance en terme de correction d'erreurs pour un nombre d'itérations données, mais le second code a une latence plus grande. Cette latence augmente avec chaque itération. Maintenant que nous avons fait un survol très rapide de la partie télécommunication, les prochaines sections couvrent les aspects d'intégration: Nous discutons notamment les méthodes de conception et les architectures reconfigurables pour donner une bonne idée de l'approche que nous avons préconisée dû aux lacunes des méthodes de conception actuelles.

2.2 Registres à Décalage

Les registres à décalage sont des circuits séquentiels qui permettent de décaler les données dans un sens ou dans l'autre tout en étant une source de stockage d'information numérique. Il existe divers types de registres à décalage. Certains sont unidirectionnels d'autres sont bidirectionnels. Les registres à décalage peuvent être implémentés différemment selon leurs applications. Les trois aspects importants à considérer sont la puissance consommée, les ressources matérielles utilisées et la configurabilité.

Sur le premier aspect, George et Alfke [13] ont décrit une implémentation d'un registre à décalage avec rétroaction linéaire (LFSR) de 15 bits qui utilise seulement une tranche d'un bloc de logique configurable avec une macro SRL16. Cette macro permet de réaliser un registre à décalage configurable de 1 bit de large par 16 bits de profond dans un module de base d'un FPGA. Alfke [1] propose une implémentation qui utilise de la mémoire à accès aléatoire (RAM) et un compteur LFSR pour simplifier la conception.

Pour le deuxième aspect, deux approches sont considérées. La première approche présentée par Lowy [18] est une structure parallèle. Cette technique permet de réduire la consommation de puissance par un facteur allant jusqu'à 3 pour une fréquence d'opération donnée.

La deuxième approche présentée par Chandrakasan [8] montre comment la technique multi-phase peut réduire la puissance par un facteur égal au nombre de phases. Par exemple, une architecture avec deux phases réduit la puissance par un facteur de deux.

Pour le dernier aspect, la flexibilité est requise pour configurer les registres à décalage sans avoir à passer par une nouvelle synthèse. La littérature est relativement pauvre en articles pour des registres à décalage configurables. Pour de petites longueurs de registre à décalage, l'utilisation de multiplexeurs est largement préconisée, telle que les SRL16 du FPGA [14]. De plus, Gigliotti [14] propose une façon efficace de créer des registres à décalage qui utilisent plusieurs interconnexions configurées dynamiquement.

Ces structures sont utilisées pour les petits et moyens registres à décalage. D'autre part, ces techniques demandent beaucoup de ressources pour implémenter les petits, moyens et longs registres à décalage lorsque ceux-ci doivent réaliser toutes les longueurs d'un registre à décalage.

2.3 Méthodologie de conception

Dans cette section, nous ne prétendons pas établir une revue de littérature élaborée des méthodologies de conception compte tenu de la richesse du domaine. Nous désirons plutôt présenter brièvement les concepts connexes à ceux que nous explicitons dans ce mémoire afin de situer notre travail par rapport aux méthodes actuelles. Dans une méthodologie de conception, on a plusieurs niveaux d'abstraction. Chacun des niveaux d'abstraction inclut des langages qui procurent des avantages et des inconvénients pour la réalisation comportementale ou matérielle d'un système. Par exemple, la conception haut niveau en utilisant Matlab, C, ou C++ donne la flexibilité permettant de faire une description fonctionnelle d'un système sans pour autant avoir les contraintes d'une implémentation physique. Dans ce sens, un design doit être fait avec un niveau

d'abstraction le plus élevé possible pour être en mesure d'exploiter les différentes implémentations possibles avec un des degrés de liberté disponibles. D'autre part, le design à haut niveau d'abstraction ne permet pas d'avoir une implémentation la plus optimale pour une spécification donnée. À l'opposé, un langage de bas niveau d'abstraction comme le RTL permet d'optimiser l'implémentation et de la rendre optimale pour une technologie donnée et un algorithme ciblé. La façon de concevoir un système dépend du niveau d'abstraction et du langage utilisé. Par exemple, prenons une spécification d'un décodeur qu'on veut implémenter. La première étape consiste à prendre un langage tel que C++ pour faire une validation comportementale. Cette phase est requise pour valider un concept. Par la suite, nous voulons l'implémenter dans une technologie donnée. Plusieurs approches peuvent être considérées, nous en considérons deux pour faire le survol des méthodologies de conception reliée à nos travaux de recherche. Nous présentons à la section suivante deux approches traditionnelles. La première approche est la conception au niveau RTL en utilisant par exemple le langage VHDL. La deuxième approche consiste à la synthèse de haut niveau. Celle-ci contient la première approche, mais de façon indirecte, car c'est fait automatiquement par un outil logiciel.

2.3.1 Conception niveau RTL

Cette méthode de conception permet de faire des implémentations matérielles. À partir de ce niveau, un outil de synthèse fait pour une technologie donnée, transforme la

spécification RTL en circuits réels. La Figure 2.4 montre le flot de conception pour réaliser ce dernier.

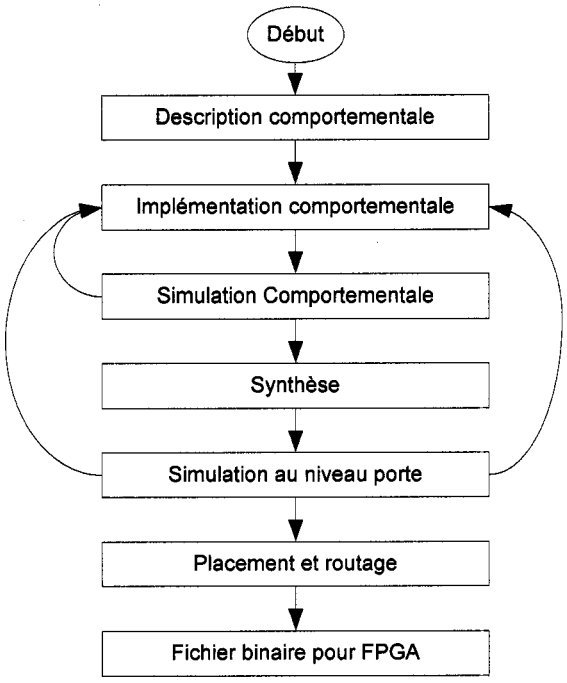

Figure 2.4 Flot de conception RTL

La première étape consiste à faire une description comportementale du système. Dans cette étape, on définit comment le système doit réagir : ceci peut être exprimé comme une équation ou une opération mathématique. Par exemple, si on veut faire E=A*B+C*D, ceci correspond à une description fonctionnelle. La deuxième étape consiste à implémenter un modèle de ce comportement en utilisant un langage RTL comme le VHDL. Ce langage permet de faire des modules et de les interconnecter ensemble en tenant compte de l'implémentation future du système. Dans notre exemple, on peut faire trois modules tel qu'illustré à la Figure 2.5.

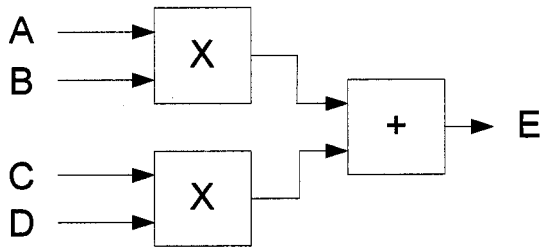

Figure 2.5 Exemple RTL

Cette figure montre comment on passe d'une spécification descriptive à une implémentation comportementale. L'équation est séparée en modules qui sont écrits avec le langage VHDL. Avec ce langage, on peut faire en sorte que les modules fonctionnent avec une horloge commune pour traiter les données à chaque coup d'horloge. La prochaine étape est la simulation comportementale. Cette étape est critique, car elle permet de simuler et de valider le modèle VHDL avec un simulateur tel que *MODELSIM*. Une fois la validation du code VHDL faite et que l'opération voulue est simulée correctement, on passe à l'étape de synthèse. Cette phase permet de prendre le code VHDL pour en générer des portes logiques. Ceci correspond au passage d'un niveau d'abstraction à un autre. Par la suite, on peut simuler à nouveau pour valider encore le concept au niveau porte. Une fois que cette étape est remplie, on passe à la phase du placement routage. Celle-ci permet de faire le placement des portes sur une implémentation réelle et de router les interconnexions. À ce niveau, on change encore de niveau d'abstraction puisqu'on passe du niveau des portes logiques à une implémentation au niveau transistor, ce qui mène à produire un fichier binaire pour configurer le FPGA. On va utiliser le VHDL, langage de bas niveau, pour faire l'optimisation de puissance et

de ressource des modules. Cette approche a été longtemps utilisée, cependant elle commence à n'être plus efficace puisque les systèmes actuels sont de plus en plus complexes. Les concepteurs demandent de plus en plus d'élever le niveau d'abstraction auquel s'effectue le travail de conception. Ceci pousse au développement d'outils.

2.3.2 Synthèse de haut niveau

Cette approche consiste à prendre des outils de synthèse haut niveau pour faire une implémentation de bas niveau comme mentionné à la section précédente. Ces outils permettent de passer d'un haut niveau d'abstraction à un bas niveau comme le niveau RTL. L'avantage de cette façon de faire est qu'un concepteur de système n'ayant pas de connaissance au niveau RTL peut produire une implémentation matérielle. Quelques avantages qui découlent de l'utilisation de la synthèse de haut niveau sont la réduction du temps de conception, la réduction des erreurs et la capacité de faire de l'exploration architecturale tel que présenté en [21]. Ce processus permet de produire plus d'un design en un temps réduit pour permettre aux développeurs d'explorer, par exemple, les compromis entre le coût, la vitesse et la dissipation de puissance. De plus, la synthèse haut niveau permet d'avoir un flot de conception auto documenté, c'est-à-dire que l'outil permet de garder les choix de design et leurs impacts. Ces méthodes améliorent l'accessibilité des technologies VLSI en réduisant les détails technologiques que les concepteurs doivent connaître. Les grandes étapes de la synthèse haut niveau sont présentées en [21] et [22]. Nous retiendrons quatre grandes étapes importantes : la compilation, l'ordonnancement, l'allocation et la liaison. La première étape consiste à

prendre la description comportementale pour la traduire en une représentation interne. Cette étape est réalisée par la transformation et l'analyse de la description comportementale pour en extraire une description graphique du comportement du système. Ceci permet d'avoir l'essentiel des opérations tout en tenant compte de la dépendance des données.

L'étape suivante consiste à faire l'ordonnancement. Cette étape permet de faire des compromis entre les performances et les coûts. Peng [22] définit trois types d'ordonnancement : constructif, global et de transformation. Nous allons voir quelques exemples pour chaque types. Les méthodes ASAP et ALAP sont des algorithmes constructifs. Le premier algorithme permet de faire les opérations le plus tôt possible, tandis que le deuxième organise les opérations pour qu'elles s'exécutent le plus tard possible. FDS est un algorithme global qui permet de balancer les opérations parallèles. Il permet de faire un graphique de distribution à partir des deux premiers algorithmes. L'objectif de ces algorithmes est d'ordonnancer les opérations pour qu'elles s'exécutent le plus rapidement possible tout en minimisant les ressources. Le dernier type d'ordonnancement est la transformation ; celle-ci s'obtient à partir d'un ordonnancement initial connu qui est graduellement transformé pour obtenir un ordonnancement final. Il existe des ordonnancements avancés qui tiennent compte, par exemple, des boucles et des opérations multi-cycle.

L'allocation est la prochaine étape. Celle-ci permet de déterminer le nombre d'unités fonctionnelles à partir du nombre d'éléments de mémoire de stockage et du nombre de

bus. Dans cette étape, nous considérons un ensemble d'éléments pour l'implémentation du circuit.

La liaison est la dernière étape, celle-ci fait le lien entre les étapes d'ordonnancement et les allocations. Cette étape a pour but de minimiser le coût matériel des registres, des unités opératoires, des bus et des multiplexeurs tout en réduisant les délais.

Les deux dernières étapes sont inter-reliées. Nous allons résumer quelques algorithmes existants présentées dans [22]. La discussion suivante présentera une méthode constructive, une formulation théorique sous forme de graphe et l'allocation par transformation. L'algorithme vorace, souvent appelé *Greedy*, commence avec un chemin de données vide et ajoute graduellement des ressources fonctionnelles pour le stockage et les interconnexions selon les besoins. Des algorithmes de formulation théorique sous forme de graphe comme le partitionnement par clique [22] permet de faire de l'allocation en faisant une représentation indirecte. Celle-ci utilise des heuristiques pour travailler sur la représentation indirecte pour faire des allocations et des liaisons.

Il existe selon [22] des concepts avancés de synthèse haut niveau selon lesquels on tient compte, par exemple, des pipelines, de la dissipation de puissance, de la testabilité et de la réutilisation des designs. Ces méthodes sont actuellement largement utilisées mais ne correspondent pas actuellement au besoin des architectures configurables. Cependant, la méthode proposée plus loin s'inspire des principales étapes telles que la compilation, l'ordonnancement, l'allocation et la liaison et pour tenir compte des pipelines, de la réutilisation des designs et de la dissipation de puissance. La prochaine section montre différente aspects de la recherche pour cette sorte d'architecture.

2.4 Architectures reconfigurables

Dans cette section, nous faisons une brève revue de ce domaine pour mettre en relief nos travaux de recherche. Les architectures reconfigurables (AR) ont pour objectif de répondre aux besoins d'applications pour lesquelles les ASIC ne sont pas assez flexibles et les microprocesseurs ne sont pas assez performants [17]. De plus, ces architectures sont un nouveau concept pour répondre aux demandes de performance et de flexibilité pour des applications données [4]. La possibilité d'avoir un système basé sur une architecture reconfigurable, pour le personnaliser dans un contexte donné afin d'avoir le bon flot de données avec le bon flot d'opérations augmente significativement les bénéfices [4]. Les nouveaux concepts utilisant les architectures reconfigurables promettent des solutions intermédiaires dans l'espace de conception en ce qui a trait au compromis entre la flexibilité et les performances. En utilisant des AR, les performances doivent être maintenues tout en accroissant la flexibilité. Dans cette section, nous résumons quelques aspects de ce domaine de recherche en nous basant sur le traitement de K. Bondalapati et V. K. Prasanna, [4]. Ces auteurs présentent cinq de ces aspects qui vont des architectures, aux modèles théoriques, aux applications, en passant par les algorithmes et les méthodes de synthèse et les outils qui les implémentent.

Les architectures présentées dans [17] et [4] sont des classes et systèmes d'architecture qui proposent des façons différentes d'organiser et d'interfacer la logique configurable. Ces architectures sont conçues avec une granularité fixe ou variable. La granularité représente le niveau auquel on configure les composantes. Par exemple, une granularité

fine permet la configuration au niveau des portes logiques, tandis qu'une grande granularité traite le problème au niveau des opérateurs tels que les additionneurs, les soustracteurs et les pondérateurs.

La granularité va affecter les réseaux d'interconnexions et la façon dont les unités sont disposées. En général, une granularité trop fine n'est pas efficace parce qu'elle demande beaucoup de ressources de routage et ne permet pas beaucoup de flexibilité pour router. Dans la majorité des cas, une grande granularité sera explicitée dans ces architectures [17] qui sont classées selon le nombre de bits par opérateur (ALU). Par exemple, une granularité de 16 bits indique que la configurabilité se fait au niveau de variables de 16 bits. Par exemple, nous pouvons avoir 4 ALU de 4 bits pour faire le 16 bits. Les architectures actuelles sont conçues pour être utilisées pour des applications spécifiques. Le Tableau 2.3 montre 5 exemples d'architectures tirés de [17].

Tableau 2.3 Architecture reconfigurable à gros grain [17]

Projet	1ième publ.	Source	Granularité	Application ciblé
PADDI	1990	[9]	16 bits	DSP
RaPID	1996	[12]	16 bits	Pipeline
RAW	1997	[23]	8 bits, granularité variable	Expérimentale
CHESS	1999	[19]	4 bit, granularité variable	Multimédia
DreAM	2000	[3]	8 & 16 bits	Sans-fil, prochaine génération

Ces architectures ont été développées dans un but précis avec des applications ciblées. À l'heure actuelle, aucune de ces architectures ne procure une solution à nos objectifs. Cependant, nous allons couvrir les autres aspects des AR. Pour utiliser ces architectures de façon optimale, on a besoin d'un modèle théorique.

Les modèles théoriques servent à prendre un algorithme et à en faire l'analyse pour configurer une architecture de façon efficace. De plus, les architectures reconfigurables sont créées et optimisées pour certaines applications tel qu'il a été montré au Tableau 2.3. En plus des modèles théoriques, pour être en mesure d'exploiter au maximum une architecture configurable, il faut aussi avoir des algorithmes de synthèse.

Les algorithmes de synthèse dans les architectures reconfigurables doivent organiser le flot de données et faire face au problème de mapping. Les méthodes actuelles sont conçues de façon générale pour des boucles dans des programmes généraux. Ceux-ci procurent des opérations répétitives et mettent l'accent sur le pipeline et les opérations parallèles tout en ciblant le matériel reconfigurable.

Présentement, les outils de mapping ne sont pas faits pour les architectures reconfigurables. Cependant, quelques outils sont en développement pour avoir des langages de hauts niveaux comme le C et faire la simulation de systèmes reconfigurables. Cela mènera à la plate-forme configurable dynamiquement.

Nous avons vu que les méthodes de conception évoluent avec la technologie et les besoins. Nous avons commencé avec une méthode de bas niveau tel que le VHDL. Ensuite, on a utilisé des outils de synthèse de haut niveau pour générer des systèmes plus complexes et ce, le plus rapidement possible. Maintenant, la forte demande pour des architectures reconfigurables afin d'avoir à la fois les performances des ASIC et la flexibilité de microprocesseur ne cesse de croître. La flexibilité et l'utilisation d'une architecture reconfigurable seront traitées pour notre cas. Pour faire le lien avec nos travaux de recherche, nous empruntons des éléments de plusieurs méthodes de

conception pour faire un décodeur reconfigurable. Le chapitre suivant propose diverses structures de registres à décalage configurables ou non, avec le VHDL, pour en réduire la puissance tout en utilisant les ressources matérielles de façon judicieuse. L'utilisation du VHDL est requis dans ce cas, car nous travaillons au niveau de la technologie. Ensuite le chapitre suivant montre certains concepts pour arriver à produire un décodeur configurable. L'utilisation des principales étapes et des concepts avancés de la synthèse haut niveau seront utilisés. De plus, les concepts d'architectures reconfigurables seront utilisés, tel que la granularité variable et les modèles de l'architecture.

CHAPITRE 3

REGISTRE À DÉCAGE GÉNÉRIQUE ET CONFIGURABLE DE FAIBLE CONSOMMATION DE PUISSANCE POUR LA RÉALISATION MATÉRIELLE D'ENCODEURS ET DE DÉCODEURS CONVOLUTIONELS

Ce chapitre présente un article, écrit en collaboration avec Y. Savaria, D. Haccoun et N. Bélanger, qui a été soumis en 2004 dans IEE Proceedings on Circuits Devices and Systems. La première section fait un sommaire en français de cet article.

3.1 Sommaire

L'article que nous présentons dans ce chapitre décrit de nouvelles méthodes pour implémenter les registres à décalage génériques configurables avec une faible dissipation de puissance. En introduction, les problématiques matérielles de l'algorithme du décodeur sont décrites telles que les pipelines qui sont prédominants et de leur accroissement selon le code de l'algorithme choisi. Par la suite, une revue de la littérature est faite. Ensuite, nous faisons ressortir l'importance de l'impact d'avoir des registres à décalage avec une consommation de puissance très faible et d'avoir des registres à décalage configurables.

De nouvelles méthodes sont montrées telle que la structure d'activation. Cette façon de faire tire profit de deux méthodes actuelles : multi phase et parallèle. Nous suggérons une nouvelle méthode pour avoir les mêmes avantages sans toutefois avoir la complexité du multi phase. Une fois que cette méthode est décrite, nous analysons son impact sur la dissipation de puissance. Par la suite, nous produisons avec cette nouvelle méthode une

structure configurable. Nous indiquons que la structure d'activation, configurable ou non, peut-être utilisée avec le multi phase toutefois avec sa complexité. Cette structure est bonne pour des registres à décalage de tailles moyennes et larges.

Pour faire des longs registres à décalage, nous avons recours aux mémoires. La méthode actuelle est un registre à décalage basé sur la mémoire. Nous proposons d'ajouter un convertisseur d'entrée et de sortie à cette structure. Celle-ci donne naissance à la structure basée sur la mémoire avec convertisseur, le but étant toujours de réduire la dissipation de puissance. En utilisant, la structure d'activation précédemment décrite comme convertisseur, nous arrivons à réduire considérablement la dissipation de puissance. La structure basée sur la mémoire avec convertisseur est ensuite convertie en structure configurable. Nous gardons toujours la réduction de puissance des nouvelles structures tout en les rendant configurables.

L'analyse de chaque type de registre à décalage qu'il soit connu ou nouveau est analysée au niveau des ressources et de la dissipation de puissance. Ainsi, nos deux nouvelles méthodes de base sont comparées avec les méthodes conventionnelles. Par la suite, nous montrons une méthode sélective pour choisir quelle structure est la plus appropriée dans un contexte donné. Nous illustrons les résultats produits par cette recherche exhaustive.

Finalement, nous comparons avec des estimateurs de puissance deux décodeurs pour plusieurs combinaisons de méthodes d'implémentation des registres à décalage. À ce niveau on utilise le langage VHDL pour faire la comparaison qui sera faite à très bas niveau.

3.2 On low power configurable and generic shift register hardware realizations for convolutional encoders and decoders

3.2.1 Abstract

Novel methods to implement low power hardware and configurable architectures comprising several different kinds of shift registers in FPGAs are presented. New approaches are also described to reduce power dissipation of shift register structures without compromising their configurability. The proposed structures are particularly effective to reduce the power dissipation of shift registers of medium and large lengths. A systematic method to select the best shift register structure is also provided. The proposed structures and the selection method are generic, and they can be configured statically or dynamically. It is shown that they are well suited for implementing powerful convolutional encoders and suitable decoders associated with forward error correction techniques, such as iterative threshold decoding.

3.2.2 Introduction

A powerful and practical forward error correction technique using specific convolutional codes called convolutional self-doubly orthogonal codes, in conjunction with a simple iterative threshold decoding technique was recently proposed by Cardinal, Haccoun and Gagnon [6], [7]. This paper explores the implementation of that error control technique using field programmable gate arrays (FPGAs). The decoding algorithm proposed in [7] can easily be mapped into hardware. This mapping produces a heavily pipelined

architecture with a large portion of its hardware composed of long shift registers (SRs), tapped at specific intervals that specify the particular code of a family of so-called systematic self-doubly orthogonal convolutional codes. The sizes of the SRs correspond to the length or memory of the code and are strongly dependent on a parameter of the encoder called J, which is the number of parity equations used. That number J determines the error correcting capability of the code. The length of the shift register of the encoder (also called span) grows approximately according to a fourth order polynomial in J [7]. In order to improve its error correcting capability, the proposed decoding algorithm operates iteratively, but the improvements tend to saturate after only a few iterations, typically 5 to 8. This is significant since the decoding cost is directly proportional to the number of iterations. The number of taps in the SRs of the decoder grows as (J*(J-1))/2 [6], [7], while their size, which corresponds to the memory of the encoder, grows as the same fourth order polynomial in J as in the encoder. For example, with J=9, the length of the largest required SR is 912, while for J=10, the required length reaches 1698 stages.

The target hardware implementation of the decoder requires various types of SRs. Some have taps while others have none. In general, a tapped register can be assembled by connecting untapped registers of lengths determined by the distance between the required taps. Untapped registers can be implemented in several ways. Moreover, when the target implementation has more than one decoder, it should include dynamically configurable SRs in order to share hardware as much as possible.

This paper explores the tradeoffs involved in assembling the best SRs to match the application's requirements. Optimality is measured in terms of the implementation

complexity, configurability and power dissipation.

In FPGAs, SRs can be implemented in different ways depending on their target use. Typically, a first concern is the minimization of the resources consumed for implementing them. George and Alfke [13] described an implementation of a 15-bit linear feedback shift register (LFSR) that uses only one slice of a configurable logic block (CLB) with the SRL16 macro. This macro is a 1-bit wide shift register configurable from 1 to 16-stages and implemented in Xilinx FPGAs. Alfke [1] further proposes an efficient implementation of a long SR using random access memory (RAM) and a LFSR counter, which simplifies the design even more, and makes the design even simpler.

A second concern of growing interest is the reduction of the power dissipated by SRs. It has been addressed by Lowy [18] where he proposed a parallel structure for implementing LFSRs. His technique reduces the power dissipation by a factor of up to 3 for a given speed of operation. On the same problem, Chandrakasan [8] shows how a multiphase architecture can reduce the power dissipation by a factor that is close to the number of phases.

Flexibility is a third concern, as there is a growing demand for hardware structures that can be configured statically or dynamically. This feature enables reuse and can significantly improve the efficiency of the design. However, until now, little has been reported about length-configurable SRs. For small length shift registers, it is common to use multiplexers to set the length of the register, like in SRL16 [14] constructed using FPGAs. In [14], an efficient multi-tap shift register is proposed. This structure is used with a medium size shift register. Unfortunately, implementing those techniques is rather

costly when a wide range of SR lengths must be supported.

The paper is organized as follows. Section 3.2.3 elaborates on the problem formulation. In Section 3.2.4, parallel and multiphase structures are proposed to reduce the power dissipation in long and medium length SRs. These structures directly target FPGA implementations, although they could also be adapted to ASIC frameworks. In Section 3.2.5, a method for choosing the best structure among a set of considered alternatives is presented and applied to six types of SRs. Moreover, the various available SR implementation styles are compared with respect to cost and configurability. Finally, Section 3.2.6 presents some concluding remarks.

3.2.3 Problem Formulation

Shift Registers are key components of our target decoder architecture. In the two reference designs discussed in this paper, using a J=10 code and a J=9 code respectively, the desired error performance is obtained with eight iterations. Each iteration of the algorithm is implemented by a dedicated pipeline stage of the decoder, where the first stage of the decoder is different from all the others [7]. Table 3.1 summarizes the SRs that are required to implement these two decoders. This table gives the number of shift registers (#SRs) that have a given number of taps (#TAPs), and their respective length and width for the two reference designs. As an example, the third line states that seven 903-stage 6-bit wide shift registers with 35 taps each are required.

Table 3.1 SRs in Two Typical Decoders with 8 Iterations

	#SRs	#TAPs	Length	Width
J=9, M=912	14	0	912	3
	1	36	912	3
	7	35	903	6
	8	8	912	3
J=10, M=1698	14	0	1698	3
	1	45	1698	3
	7	44	1696	6
	8	9	1698	3

Reading from Table 3.1, the total number of bits to shift is roughly equal to 111*M, where M is the length of the code. Each SR with taps is assembled by connecting untapped registers of lengths determined by the distance between the required taps. The number of shift registers grows as 8*(((J*(J-1))/2) +J)-1. Hence for J=9, 100854 SR bits in 359 untapped SRs are needed, whereas for J=10, that number grows to 188394 bits spread over 439 SRs. Clearly, high density and low power SRs are highly desirable for implementing the considered decoder. Moreover, some applications, such as conducting research on convolutional coding and iterative threshold decoding require that we merge codes together to enable a quick switching from one to the other. For those applications, configurable shift registers are clearly very advantageous. The remainder of the paper presents an effective implementation methodology for shift registers of different sizes and widths that are critical for our reference designs.

3.2.4 New Low-Power Shift Register Structures

In this section, two structures are proposed to reduce the power dissipation in SRs. The first is based on a parallel, multiphase memory-based structure. The second structure is designed to reduce power dissipation when longer SRs are needed. The flexibility of those structures, designed to be dynamically configurable, will be demonstrated.

3.2.4.1 Shift Register Power Reduction Methods

This section reviews the parallel multiphase technique and proposes the selective activation method to reduce SRs power dissipation. The second part of this section will demonstrate the flexibility of the selective activation method with regards to configuration.

3.2.4.1.1 Parallel Multiphase Technique

A parallel multiphase implementation uses multiple clocks to selectively activate SR sections [18]. The goal is to reduce the switching activity. It can be applied to small, medium and long SRs. For example, an N-stage SR can be controlled by four phases, each driving one of four registers of length N/4. Thus, to implement a SR that appears to work at 100 MHz, a 25 MHz multiphase parallel structure would be used. The two structures are functionally equivalent, but the activity (and power consumption) of the multiphase implementation is reduced by a factor that is close to four. Note that the reduction in power consumption partly comes from a reduced load on the clock distribution network (i.e. only one fourth of the load is driven at each clock cycle). In

FPGAs, this technique is constrained by the number of clock networks and the power reduction is obtained at the cost of an increased complexity.

3.2.4.1.2 Proposed structure

A structure that leverages the advantages of multiphase operation without its complexity is now proposed. An example is shown in Figure 3.1 for an N-stage SR. In this example, a two-bit wide structure is driven by four enabling strobes. Each column handles one of the bits of the data items to be shifted, and each row corresponds to a time strobe. Here, the basic SR modules have a length of N/4. These modules can be based on SRL16, flip-flops or memories. In the first time interval, the first row buffers are enabled, in the second time interval, the second row buffers are enabled, and so on. This scheme can be generalized to M rows and M strobes, where M is any positive integer.

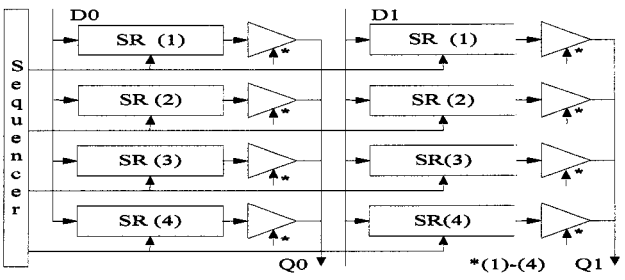

Figure 3.1 Selective activation method

An M stage one-hot sequencer produces the desired strobes. In order to support register lengths that are not a multiple of four, the output has to be skewed with regards to the input. This means that the strobe used to select the output bit is the input strobe delayed by the required number of clock cycles.

3.2.4.1.3 Impact on Power Dissipation

The proposed structure adds little to the complexity of the SR, while reducing very significantly the activity. Activity is reduced by a factor close to M (number of strobes). Thus, the power dissipation is reduced by a similar factor albeit slightly smaller because the activity on the clock distribution network is not reduced. Alternately, as described in Section 3.2.4.1.1, selective activation can be used in a multiphase implementation in order to further reduce the power dissipation, but at a higher implementation cost.

The tri-state bus connections in Figure 3.1 could be replaced by conventional multiplexers. This structure works best for medium length, wide SRs. The next subsections describe a configurable version of this structure while section 3.2.4.2 proposes an approach more suited for long SRs. Also, it is shown in section 3.2.5 which structure has the lowest power dissipation for a given SR length.

3.2.4.1.4 Configurable Selective Activation Method

There is a growing interest for configurable architectures where a single design can serve multiple applications. Dynamic configurability also allows changing functionality to respond to changing needs. This section shows how to make the selective activation architecture configurable while keeping its power reduction properties. It does so by reusing components as much as possible, as opposed to replicating them. Figure 3.2 shows how to implement the generic configuration. As will be shown, the proposed structure is very flexible since it provides more than one way to implement the same target SR length. This feature allows making complexity and power efficiency tradeoffs

as explained later. Note that each SR is terminated by a flip-flop to make it fully synchronous, which helps reaching high performance. Also note that configuration using this technique can only be done off-line. This structure is configured using 3 registers providing respective control words C0, C1 and C2, as shown on the figure.

This architecture is used as follows. A series of sub-SRs (SR(1) to SR(M')) form the greater part of the SR, and their outputs drive the input of another sub-SR: SR(M). As with the selective activation method, each sub-SR is shifted (and its output used) according to the state of a one-hot sequencer (located on the left of Figure 3.2) that is implemented with the C1 register, composed of flip-flops (identified by the boxes labeled D on the left of the figure) and of a multiplexer controlled by the C1 control word. If the total length of the SR is not a multiple of the number of sub-SR (i.e. M' in the figure) then the last sub-SR (SR(M)) is used to complete the SR.

The three configuration registers, C0, C1, and C2 respectively control the length of SR(M), the number of sub-SR used, and the length of those sub-SR. Let us define M as the total number of SR rows, and M' as the number of rows controlled by the sequencer (M'=M-1). The configured number of rows controlled by the sequencer is defined as Mv'. In Figure 3.2, N is the word length of the data items to be shifted and thus, it can be seen that each sub-SR column handles one bit of the data items.

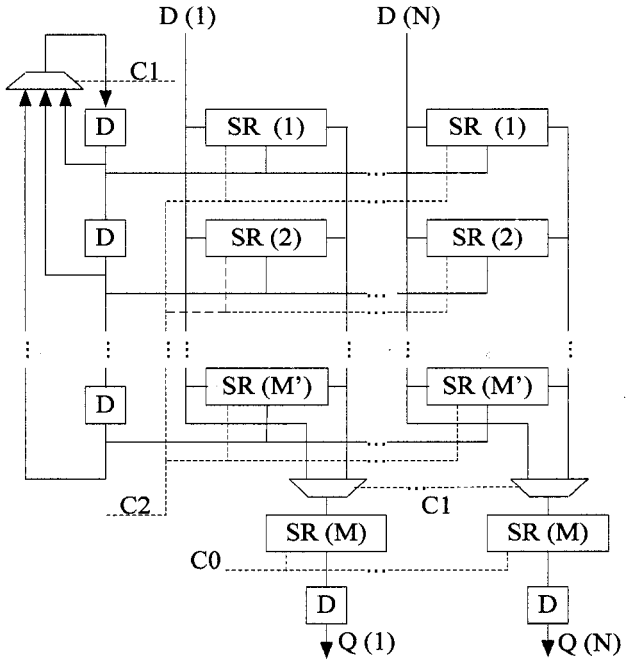

Figure 3.2 Configurable selective activation method

The physical length of each sub-SR is defined as Lm' and the configured length used in a given context is Lv'. Note that, in order to reduce the implementation cost and to avoid unnecessary redundancy, all the sub-SRs controlled by the sequencer have the same configured length. Moreover, the last sub-SRs row (i.e. not controlled by the one-hot sequencer) has a maximal size Lm and a configured length Lv. The minimum length of this configuration is 2 (because the data has to go through the SR(M) sub-SRs and the final flip-flops at the bottom of the Figure 3.2). In consequence, the maximum length is M'*Lm'+Lm+1. The length of the SR, when configured, is defined as A and is computed as Mv'*Lv'+Lv+1. Configuration registers C0, C1 and C2 affect respectively Lv, Mv' and Lv'. This architecture is generic. It is also redundant as some lengths can be configured in different ways as shown in Figure 3.3, which shows the histogram of the

number of ways to configure the different possible SR lengths when changing only the values of C1 and C2. The total redundancy is even higher than shown in Figure 3.3 when C0 is considered, but as is apparent from Figure 3.3, some SR lengths cannot be obtained if it is configured only through C1 and C2. However, controlling only C0-C1 or only C0-C2 allows producing all possible lengths.

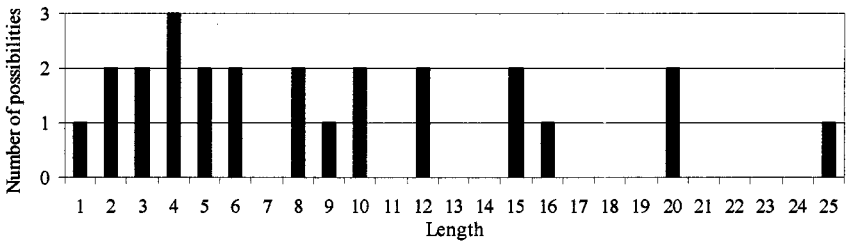

Figure 3.3 Distribution of possibilities vs. length for M'= Lm'=5

Indeed, when C0 and C2 are used, if Lm is equal to M' (the stride caused by an incremental change in C2) all possible lengths can be obtained. Similarly, when C0 and C1 are used, Lm should be equal to Lm' to enable producing all possible lengths without redundancy. Thus, this architecture is generic and it can be easily configured to any length through C0 and C1 or through C0 and C2.

When more than one set of parameters gives the desired SR length, the choice of the best control parameters set should be done according to user defined objectives, such as decreasing power dissipation or reducing hardware complexity. Depending on the technology used for the implementation, the sub-SR power dissipation can be constant or it can change following the configured length of the sub-SRs. For example, an implementation based on flip-flops has a power consumption that changes with the length of the SRs, while an implementation based on memories has constant sub-SR power

consumption. Thus, when the power dissipation of the sub-SRs is variable, C0 and C2 should be used in order to minimize the power consumption of the sub-SRs. On the other hand, when their power dissipation is constant, any pair can be used, because the total power consumption is constant since only one sub-SR is activated at a time per column. But if hardware complexity is an important criterion, C0 and C1 should be used because, in this situation, the sub-SRs do not need to be configurable, thus reducing hardware complexity and avoiding routing of C2.

3.2.4.1.5 Multi-phase Configurable Selective Activation Method

As mentioned in section 3.2.4.1.1, the proposed configurable selective activation structure can be combined with a multiphase clock. For example, a system with two phases may divide the shift register in two sections as shown in Figure 3.4.

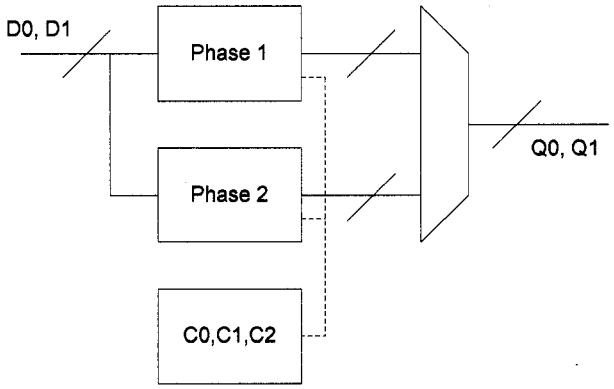

Figure 3.4 Multi-phase configurable selective activation method

If A is the total target length of the SR, we can design two SRs of length $\lfloor (A+1)/2 \rfloor$ using the method of section 3.2.4.1.4 to adjust the size of each SR by configuring parameters C0, C1, and C2. If A is odd, output data must be read from the phase 1 SR when input

data is being written into the phase 2 SR, and vice versa. If p>2 phases are available, the proposed architecture can be generalized with p SRs of length $\lfloor (A+p-1)/p \rfloor$, complemented by a controller that will skew as required the reading and writing operations. This method reduces power dissipation by a factor close to the number of phases (as explained in Section 3.2.4.1.1). The cost of this structure is slightly higher than the cost of the structure in Figure 3.2. It may be worthwhile to note that this structure can be transformed into a multi tap SR but it introduces several design constraints and their analysis is beyond the scope of this paper.

3.2.4.2 Memory-based Shift Registers

This section investigates effective methods to implement SRs based on memory. A method to reduce their power dissipation is also proposed. The objective is to better support long SRs. This is significant as it will be shown in Section 3.2.5 that memory-based SRs have the lowest power dissipation.

It is well known that long SRs are advantageously implemented using memories addressed by pointers. At each clock cycle, one data element is read out, while a new data element is written in its place. Power dissipation can be reduced because in a long SR, most of the data in the memory not involved with the words written or read do not move. However, power is dissipated by the required peripheral logic and counters used to implement the required pointers. Again, power dissipation can be reduced by lowering the activity. Later in this section, we show how power dissipation can be further reduced using data format converters that reduce activity. This concept applies to ASIC and

FPGA implementations, but it is particularly useful in FPGAs where fixed size embedded RAMs supporting selected word widths (1, 2, 4, 8 and 16) are available. For example, a straightforward implementation of a 1698 stage 6-bit wide SR could use 3 2-bit wide memories of 2048 words. Alternately, 3 8-bit wide memories of 512 words would behave identically when combined with suitable format converters on input and output as shown in Figure 3.5.

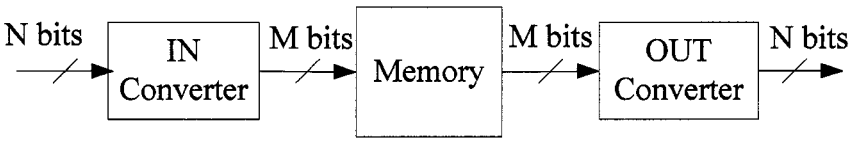

Figure 3.5 Example of memory with converter structure

The input converter transforms N-bit words into M-bit words, while the output converter transforms M-bit words into N-bit words. If M>N, the memories can be activated less often than the input word rate, thus reducing the power consumption. Here, M/N is the conversion factor, called F, which determines memory activity and power dissipation. For instance, if one wants to implement a 6-bit wide 1698-stage SR using a 24-bit wide memory, activity in the memory could be reduced by a factor of 4. However, input and output converters are needed. Analytical expressions were developed to determine, amongst the considered configurations, the one that minimizes power dissipation for a given shift register size.

Two types of converters called type-1 and type-2 are considered. A type-1 design at the input is a classical serial to parallel converter used on each bit of the N data items while, at the output, a parallel to serial converter is used. In a type-2 design, a variation of

Figure 3.1 selective activation structure with 1-deep SRs (i.e. a latch) is used as shown in Figure 3.6. This figure gives an example of input and output converters with N=2, and F=4 (thus, M=8).

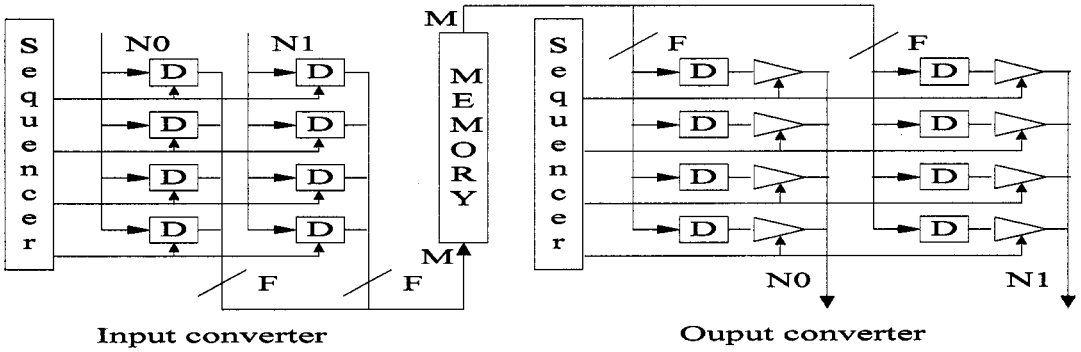

Figure 3.6 Example of a memory based SR with input and output converters

Figure 3.7 shows the impact on power consumption of the two types of converters, when three memories of variable widths are used to implement a 6-bit wide SR. With 3 memories, we have 6/3=2 and all even memory widths can effectively be used.

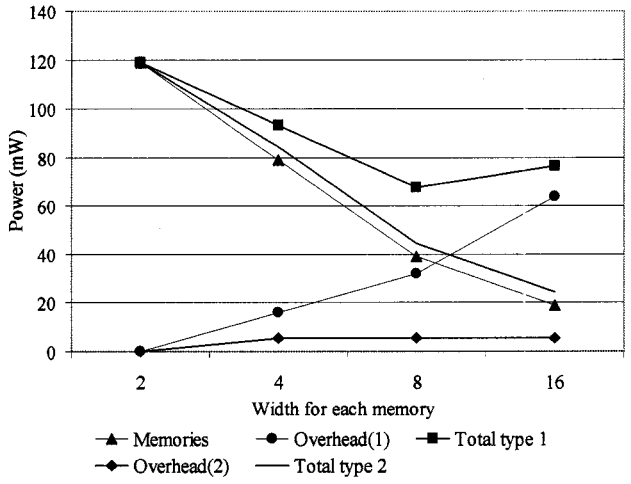

Figure 3.7 Power dissipated by the memory-based SR structure with converters implementing a 6-bit wide SR of length 1698

This figure shows that the type-2 converter allows using much larger M/N factors, which

can drastically reduce both activity and power consumption.

3.2.4.2.1 Configurable Memory based Shift Registers with Converters

This section explores how to configure memory-based shift registers while keeping their power reduction advantage. Configurability is obtained with pointers as illustrated in Figure 3.8. The main idea of this architecture is to provide a configurable shift register based on memory. As shown, two converters are needed to reduce power dissipation. Also, in order to make this structure configurable, a length configuration register is added as shown in Figure 3.8.

This SR works as follows: data is written in memory using a write counter. That counter wraps around when it reaches the end of the memory. Data is read using a read counter, which starts when the write counter reaches a value given by the length configuration register. Note that data is written to and read from memory once every F clock cycles thus, is order to support all SR lengths below the maximum imposed by the capacity of the memory, there must be a skew between the read and write cycles. This is achieved using a counter that delays the read enable pulse with regards to the write enable pulse.

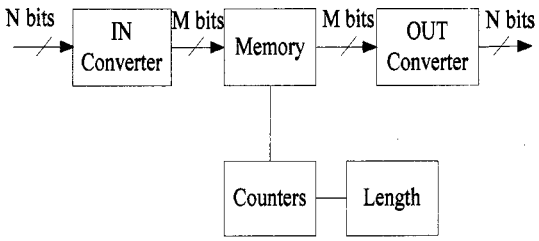

Figure 3.8 Configurable memory-based shift registers

We can also transform this structure into a multi-tap SR, but the possibilities are highly constrained, and the lengthy analysis required to describe these constraints is beyond the scope of this paper.

3.2.5 Architecture Analysis and Method to Minimize Power Dissipation

This section presents results that show the significant reduction in power consumption that can be obtained with the proposed selective activation and memory based SRs. We focus on the non-configurable structures to compare them to the traditional approaches. Also, a method to select the architecture with the lowest power consumption is described.

3.2.5.1 Analysis of Various Shift Register Structures

Six different methods for implementing a 1698-stage long 6-bit wide SR have been analyzed and characterized. All configurations can operate at a frequency of 100MHz in a VIRTEX 2000-E FPGA. That frequency was used in all power estimations reported in Table 3.2. The first four methods are known, whereas the last two are novel methods proposed in this paper. The reported estimates are worst-case power consumption on a VIRTEX 2000-E FPGA and are based on data extracted from the Xilinx power estimation tools [24]. Two activity levels were considered in order to have the best and worst power consumption cases. The activity levels chosen are 6% and 55%, because it was shown in [26] that an average design has an activity level in the 6%-12% range, while a 16-bit LFSR counter has an activity level of 55%. Cells in the Table 3.2 are merged when the activity has a negligible impact on power dissipation.

Table 3.2 Resource and Power Dissipation for a 6-bit Wide, 1698 Bit Long Shift Register implemented in a VIRTEX 2000-E FPGA

Structure	**# unit** Approx.	**Resource**	**6% activity** (Watt)	**55% activity** (Watt)
Flip-Flop	10188	26.6%	0.244	2.24
SRL16	642	1.7%	0.038	0.349
Distributed RAM	678	1.8%	0.368	
Block RAM	3	3/160 memories	0.079	
SRL16 based Sel. act.	695	1.8%	0.091	
Block RAM with converters	3	3/160 memories + overhead	0.035	

Figure 3.9 shows the impact of the power dissipation of each structure for six-bit wide registers as a function of the register length.

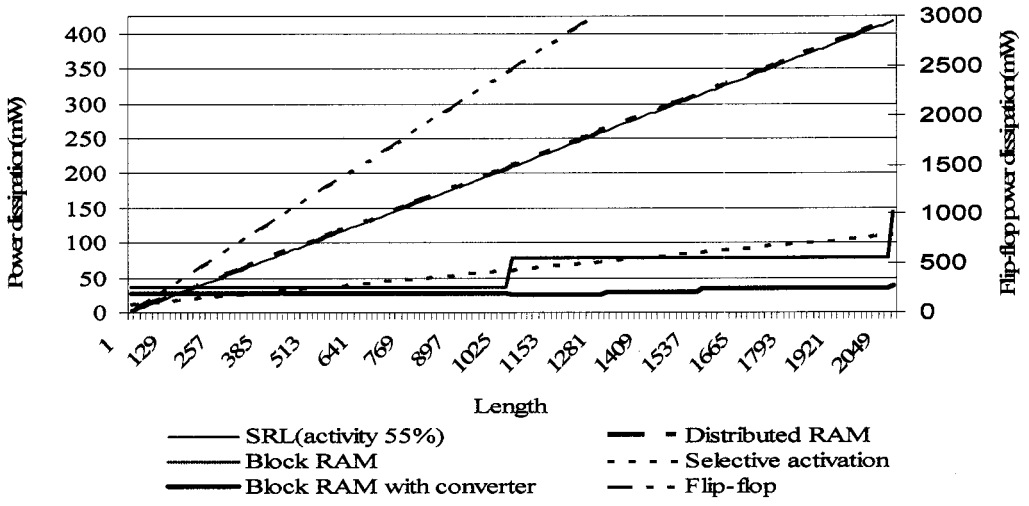

Figure 3.9 Power dissipation for various 6-bit wide SRs

This figure shows that power dissipation increases with the length of SRs. The flip-flop based structure is shown on a second vertical axis (on the right) for readability, since its dissipation is much larger than for the 5 others most of the time.

3.2.5.2 Method to Select the Structure with the Lowest Power Dissipation

The method used to find the structure with the lowest power dissipation considers five parameters to compute this value: desired throughput, register length, activity level, register width, and the threshold on power dissipation gain. The last parameter is used only with regards to the structure based on Block RAM with converters. The power dissipation of this structure is strongly influenced by the conversion factor, F, thus a search must be performed to find its lowest power dissipation. So, the last parameter is the threshold on the gain in power dissipation below which the search on F is stopped.

The dissipation of each configuration was modeled analytically and calculated for various SR configurations using the 55% worst case activity hypothesis. Our power estimation program takes into account all FPGA components required to implement each configuration using the 6 considered approaches. The configuration that produces minimal dissipation is retained. Table 3.3 shows typical results produced by the automated analysis procedure and Table 3.4 shows the hardware complexity for each structure.

Table 3.3 Summary of Best SR Structures that Minimize Power Dissipation in VIRTEX-E Technology at 100 MHz

Length	Structure	Power (mW)
3 bit wide		
1-3	Flip-Flop	0.6-1.98
4-48	SRL16	2.8-5.5
49-690	SRL16 based sel. act.	5.2-21.4
691-4096	Block RAM with converter	22.2-23.4
6 bit wide		
1-3	Flip-Flop	1.3-3.9
4-33	SRL16	4.5-7.8
34-370	SRL16 based sel. act.	9.9-26.3
371-4096	Block RAM with converter	27.8-43.2
8 bit wide		
1-3	Flip-Flop	1.7-5.2
4-33	SRL16	6.1-13.0
34-242	SRL16 based sel. act.	14.8-26.5
243-4096	Block RAM with converters	29.0-52.9

Table 3.4 Resource, Speed and Complexity for 4 SR Structures

Structure	**Complexity** (# unit)[1]	**Speed** (MHz) typical
Flip-Flop	(W*L)/4 CLBs	250
SRL 16	(W*L) / (16*4) CLBs	150
SRL16 based sel. act.	(W*L)/(16*4)+L/(16*4) CLBs	100-200
Block RAM with Converters	$\lceil (W*F)/ W_m \rceil$ Mem. +((W*F+F)/2) CLBs	150

[1] Where W is the SR width, L is the SR length, F is the conversion factor, and W_m is the memory width.

Table 3.5 shows the total dissipated power for the registers needed by the 2 typical decoders described in Table 3.1.

As in [25], we suppose that a manufacturer's goal is to design a system that has a junction temperature of at most 85°C given an ambient temperature of 45°C. For a XCV2000E-BG680, the rated power would be about 3.5 Watts without any heat sink. With forced air and a suitable heat sink, the rated power is up to 20 Watts [25]. Note that some reported powers are significantly larger than what can safely be dissipated by the device without heat sink. Clearly, low power SRs are highly desirable for implementing the considered decoders.

Table 3.5 Total Shift Registers Power Dissipation for Two Typical Designs

Set of structures used	**J=9**	**J=10**
1) Flip-flop and SRL16	4171 mW	7312 mW
2) Flip-Flop, SRL16 and Block RAM	3357 mW	4425 mW
3) Flip-Flop, SRL16, SRL16 based sel. act.. and Block RAM with converter	2427 mW	3374 mW

The first column of Table 3.5 lists 3 different ways considered to implement our two typical decoders. The first method is a straightforward implementation using only FF and SRL. It dissipates 4.17W for a J=9 decoder. When Block RAMs are used, the dissipation goes down to 3.36W. However, when we leverage all available techniques and use them optimally, the dissipation reduces to 2.43W. For the J=10 decoder, respective dissipations are 7.31W, 4.42W, and 3.37W. The best solution dissipates only 46.2% of the power required with the basic approach. Also, the novel techniques proposed in this paper allow reducing the power dissipation to only 76.3% of the power required when optimally applying the best previously available techniques. As a consequence the SRs of our two decoders could now operate without heat sinks.

3.2.6 Conclusion

In this paper, we have investigated effective means of implementing shift registers suitable for convolutional self-doubly orthogonal encoders and iterative threshold decoders. Two novel methods to reduce the power dissipation were proposed: the selective activation method and the memory based method with converters. These methods introduce modest hardware overhead, and lead to significant reductions in dissipated power. Moreover, the suggested shift registers structures are configurable while keeping their power dissipation advantages. The six considered methods were implemented in a generic way as a function of shift register length and width, and their resulting power dissipation was characterized. The SR structures among that set that minimize dissipation were characterized and tabulated.

Based on those results, we have automated the process of selecting the SR implementation with the lowest dissipation in a generic design as a function of key architectural parameters. This is a necessary step to get a generic hardware implementation of iterative threshold decoders for convolutional self-doubly orthogonal codes with reasonable power dissipation.

CHAPITRE 4

MÉTHODOLOGIES DE CONCEPTION POUR SÉLECTIONNER ET IMPLÉMENTER DES DECODEURS CONFIGURABLES

Ce chapitre présente un article, écrit en collaboration avec N. Bélanger, D. Haccoun et Y. Savaria, et qui a été soumis en 2004 dans IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. La première section fait un sommaire en français de cet article. Ce chapitre est dans une suite logique du précédent chapitre. À partir des registres à décalage configurables de faible dissipation de puissance, nous pouvons montrer avec de nouvelles méthodes que la réalisation à plus haut niveau de décodeur configurable est réalisable.

4.1 Sommaire

Le deuxième article que nous présentons dans ce chapitre couvre des méthodes pour choisir et implémenter des décodeurs. Cet article comporte les grandes caractéristiques d'un outil de synthèse de haut niveau. Nous y présentons des concepts tirés de la synthèse de haut niveau et des architectures configurables. En introduction, les problématiques matérielles de l'algorithme du décodeur sont décrites telles que les pipelines qui sont prédominants et de leur accroissement selon le code de l'algorithme choisi. Par la suite, une revue de la littérature est faite sur les méthodes de conception tel que, celle que nous avons réalisée plus tôt à la Section 2.3. Ensuite, nous faisons ressortir les grandes caractéristiques du décodeur telles que les pipelines et les additionneurs.

En utilisant C++ comme langage de haut niveau, nous montrons que nous sommes capables de faire des décodeurs configurables en faisant abstraction de la technologie. Comme dans la synthèse de haut niveau, nous présentons sous forme déjà faite, la phase compilation. Cette phase consiste à prendre l'algorithme et à le transformer. Dans notre cas, nous prenons un modèle du décodeur qui a été fait. À partir de ce modèle, nous décrivons une nouvelle façon de représenter le décodeur sous forme de zones. Notre but est de pouvoir faire un décodeur configurable. Nous expliquons que les zones de pipeline peuvent être représentées en listes. Chaque élément de la liste possède un ID unique. Ce ID est décrit comme un élément qui fait le lien entre les zones et sert pour la fusion des décodeurs. De plus, nous élaborons une autre zone qui contient les unités opératoires. Celles-ci représentent un réseau de logiques combinatoires. La notion de groupe est aussi créée pour réduire considérablement la complexité de la fusion. Ainsi, nous séparons un modèle du décodeur en zones. Par la suite, nous discutons de la fusion des décodeurs. Nous expliquons comment fusionner les zones combinatoires. Par analogie, la description de la méthode de fusion des pipelines est décrite. Le plus gros décodeur sert à créer l'architecture configurable. Ainsi, ce dernier permet de fusionner d'autres décodeurs sur celui-ci, ce qui donne une utilisation judicieuse des ressources. Comme dans tous les outils de conception, l'exploration des designs est importante. Par conséquent, nous introduisons certaines métriques pour l'exploration de plusieurs décodeurs séparés et des décodeurs fusionnés. Nous estimons les coûts et les gains reliés à la fusion des décodeurs pour produire des décodeurs configurables. Pour ce faire, nous présentons l'exemple d'une analyse de quatre décodeurs séparés et faisons l'analyse de 4 fusions différentes.

4.2 Design methodologies to select and implement configurable decoders

4.2.1 Abstract

This paper presents methods to select and to implement configurable decoders. A configurable decoder is suitable to improve the decoding process in a multi environment channel and for research applications. This type of decoder is composed of more than one decoder and its purpose is to efficiently implement forward error correction techniques, such as convolutional coding and iterative threshold decoding. We explore the possibility of merging several decoders and how to select the best configuration for a given context. We explore decoders individually and together using estimators in such a way to avoid the need to perform synthesis of the different decoders. This provides to the designer a resource and power dissipation estimate for an implementation of one decoder or of a configurable decoder. We show how to design a heavily pipelined configurable decoder while being able to optimize for resources and power dissipation.

4.2.2 Introduction

A powerful and practical forward error correction technique using specific convolutional codes called convolutional self-doubly orthogonal codes (CSO2C) in conjunction with a simple iterative threshold decoding technique was recently proposed by Cardinal, Haccoun, and Gagnon [6], [7]. This paper focuses on implementing the technique using field programmable gate arrays (FPGAs). The decoding algorithm proposed in [7] can easily be mapped into hardware. This mapping produces a heavily pipelined architecture

with a large portion of its hardware composed of long shift registers (SRs), tapped at specific intervals determined by the characteristics of the code being actually implemented. The tap positions specify the particular code used. The size of the SRs depend on the code, and is strongly influenced by the J parameter of the encoder, which is the number of parity equations used, and which determines the error correcting capability of the code. The length of the encoder grows approximately according to a fourth order polynomial in J [7]. The algorithm operates iteratively. Its error rate improves with each iteration, but that improvement tends to saturate after a few iterations, typically 5 to 8. However, the cost of decoding is directly proportional to the number of iterations. The number of taps in the SRs grows as (J*(J-1))/2 [6], [7], while their size, which corresponds to the memory of the encoder also grows significantly with J. For example, with J=9, the length of the largest required SR is 912, while for J=10, the required length reaches 1698 stages.

The target hardware implementation of the decoder requires two types of SRs: with and without taps. In general, a tapped register can be assembled by connecting untapped registers of length determined by the distance between the required taps. Untapped registers can be implemented in several ways. Moreover, when the targeted system implementation has more than one decoder, it may be attractive to create a configurable decoder in order to minimize the required hardware.

This paper describes methods to select and to implement configurable decoders. The decoding performance of a decoder is a function of its code. Different codes with the same value of J yield identical error decoding capability. Thus, the first concern is to

explore which decoder is the best for the purpose at hand and which set of decoders is better matched because each code used affects the architecture and the size of the decoder and the configurable (merged) decoder. The second concern is that we have to merge more than one heavily pipelined architectures and that the high level synthesis and the RTL-level design flow do not provide us with a methodology to design configurable architectures [21], [5]. Moreover, the reconfigurable architecture described in [17] does not yield a heavily pipelined architecture thus we need to devise a new one. Algorithmic synthesis for dynamically reconfigurable architectures gives rise to new classes of problems such as mapping computations onto the architecture [4]. The tools used to produce them rely on CAD based mapping techniques. Also, there are several tools being developed to address run-time reconfiguration [4]. Those methods are under development and do not provide a satisfactory solution. Consequently, in this paper we propose a method to design a heavily pipelined configurable decoder. The first step of this method consists of representing the decoders using a model that allows decoder merging. The second step is to merge the needed decoders following their representation under this model. Finally, the last step in our design methodology is to generate all the necessary files for the implementation of configurable iterative decoder for CSO2C.

This paper is organized as follows. The problem addressed in the paper is formulated in Section 4.2.3. In Section 4.2.4, an abstract decoder model is presented. It shows how we represent a decoder in order to have the flexibility to merge and optimize it. In Section 4.2.5, a method to merge decoders is presented along with the description of the abstraction level which is useful to do the merging. Moreover, we show how to optimize

resources and power dissipation. Finally, Section 4.2.6 presents some concluding remarks.

4.2.3 Problem Formulation

Shift Registers are key components of our target decoder architecture. Table 4.1 shows an example of two decoders each implemented with 8 iterations. Each iteration is implemented by a dedicated pipeline stage in the considered architecture. This table illustrates the number of shift registers (#SRs) that those two implementations of our reference design have. Note that some of the SRs have taps (#TAPs) while others do not. As an example, the third line states that this implementation of a J=9, M=912 decoder comprises 7 903-stage 6-bit wide SRs with 35 taps each.

Table 4.1 SRs in Two Typical Decoders with 8 Iterations

	#SRs	#TAPs	Length	Width
J=9, M=912	14	0	912	3
	1	36	912	3
	7	35	903	6
	8	8	912	3
J=10, M=1698	14	0	1698	3
	1	45	1698	3
	7	44	1696	6
	8	9	1698	3

Moreover, the decoder requires a lot of combinatorial logic such as add-mins, ponderators and adders. A ponderator is used to multiply its input by a coefficient that is between 0 and 1 (including 1). The add-min operation produces as the minimum of the absolute value of its two inputs, where the sign of the results computed as -sign(input A)sign(input B). For some applications, several heavily pipelined decoders are needed in

order to compare their performance level or to adapt the performance level to the changing environment. In those situations and considering the amount of resources required by each decoder, there is a need to produce a configurable decoder in order to reduce the hardware requirements. A method to merge decoders is therefore highly desirable thus leading to the use of configurable decoders. Consequently, the remainder of the paper presents methods to explore and to implement configurable decoders.

4.2.4 Methods to Represent Decoders

This section shows how to represent decoders in order to get the flexibility to merge them. The methods that we present are implemented in a tool we have developed. This tool is written in the C++ language.

4.2.4.1 Overview

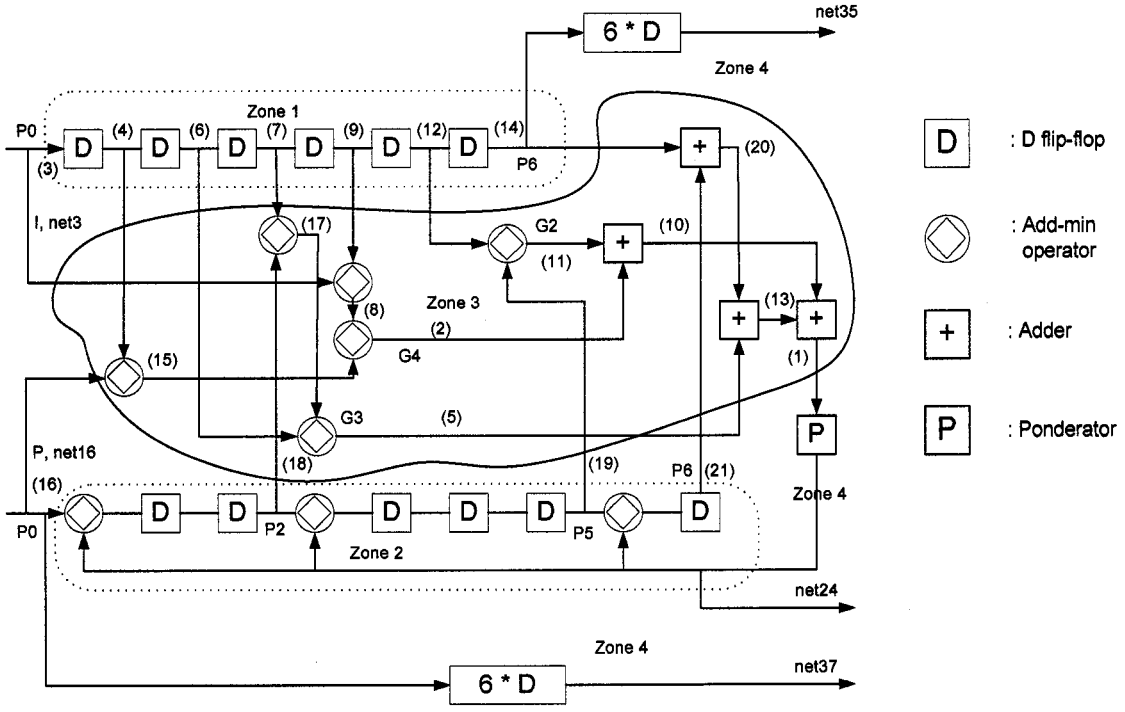

Figure 4.1 First iteration of the decoder

A decoder is composed of iterations. Figure 4.1 shows the first iteration of the reference design with a J=4 code. For illustrative purposes, this paper focuses on the first iteration but the methods are applicable to all iterations. The symbols D, +, P and the rhomb inside a circle represent respectively a flip-flop, an adder, a ponderator, and an add-min. Moreover, Gn (where n = 2, 3, or 4), Pm (where m = 0, 2, 5, or 6) and (q) (where q = 1 to 21) represent respectively a group, a position and an ID of a TAP or a component as described later. The add-min operators are assembled in groups according to the decoding algorithm in [6] (although the term "group" is only used here); this eases decoder merging as will be shown later. Each iteration is divided in 4 zones; zone 1 and zone 2

are mainly shift registers. The first zone is a SR with J*(J-1)/2 taps. The second zone has J taps and it comprises some add-mins. Two types of nodes are defined: the first type, called ITAP, is used to connect the zone 1 to the zone 3 and the second, called PTAP, is used to connect zone 2 to the zone 3. Zone 3 is the logical structure that implements most of the calculations such as add-min and adder. This logical structure is combinatorial. It could be made sequential (pipelined) in order to reduce delays but this capability was not implemented in our tool because maximizing the clock frequency was not a primary goal of this project (left for future research). These zones are defined to represent an internal view of each decoder in our tool. Moreover, these zones will be used to merge decoders. Zone 1 and zone 2 are two kinds of pipelined structures: information and parity. The expression "pipelined structure" is used to mean a design with sequential operations and composed of shift registers. Zone 3 is a logical structure representing the J equations, while zone 4 has a different structure for each iteration and it basically includes two untapped SR and a ponderator. Note that all the J taps from zone 1 and zone 2 are used in zone 3. To illustrate the methods and in order to keep the example reasonably small, the example that follows uses J=4 with the code 0, 1, 4, 6.

Section 4.2.4.2 illustrates how the heavily pipelined structures are represented in our tool while Section 4.2.4.3 shows how we model the logic structure. Finally, the global representation is explained in Section 4.2.4.4.

4.2.4.2 Heavily pipelined structure

As previously mentioned, zones 1 and 2 are heavily pipelined structures. In our tool, those zones are represented by lists. A list gives the flexibility to represent the pipeline. The information stored in those lists describes the taps of the SRs of zones 1 and 2.

Table 4.2 Information

Position	Group	Nbit	ID
0	4	3	3
1	4	3	4
2	3	3	6
3	3	3	7
4	4	3	9
5	**2**	**3**	**12**
6	1	3	14

Table 4.3 Parity

Position	Group	Nbit	ID
0	4	3	16
2	3	3	18
5	2	3	19
6	1	3	21

As illustrated in Table 4.2 and Table 4.3, we have two lists, the first one is the information data and the second one is the parity data which represent respectively zone 1 and zone 2. For example, for J=4, the list of zone 2 contains a SR with add-min in position 0, 2, 5 and 6. This is illustrated in Figure 4.2. As explained earlier there are two types of nodes: ITAP and PTAP. All ITAP form the information list (Table 4.2) and all PTAP form the parity list (Table 4.3). Each element of the lists contains three data items: the group number, number of bits and ID of the node. The group number determines the type of resource (add-min or adder) to which the tap is connected. For each TAP, a group

number of 1 actually means that the corresponding tap is connected directly to an adder, while the other values are used for the different groups of add-min. The groups represent how to connect the taps. A group 3 connects the tap to an add-min with 3 ports which can be done by 2 add-mins with 2 ports each. The Nbit item determines the size of the tap and the ID item is needed to map the tap to the logic structure and is needed to merge decoders. For example, in the information list, the sixth line, where elements 5, 2, 3, and 12 (in bold in the table), represent a 3-bit-wide ITAP which is connected to an add-min of group 2. This ITAP node has an ID of 12 which corresponds to the ID of the information list. The list grows with the number of taps. For example, for J=10, 46 elements are needed for the information list. We will further discuss the ID item in Section 4.2.4.3, 4.2.4.4 and 4.2.5. For real applications, we need resources similar to those of our reference design as illustrated in Table 4.1.

4.2.4.3 Logical structure

The logical structure represents the combinatorial logic that performs the calculation algorithm and is implemented as a tree. Figure 4.2 illustrates the concept. This graph is generated by our tool in the **dot** format [2]. Each node is defined by its type (Adder, Addmin, ITAP, PTAP), the width of the input, and the group to which it belongs. This format is used to create a graph representation that we use to show the logical structure in a way that is easy to understand. This graph represents the dataflow graph for the first iteration of the decoder. In this tree, there are sub trees where all the components are of the same type. Those sub trees are balanced to minimize the number of output bits of the

adders. In particular, the root of the tree is always an adder sub tree that is fed by sub trees of add-min components. Also, the leaves of the tree are all of type ITAP or PTAP. The tool can estimate the number of bits that we need [20]. The width of the output of an add-min is always the width of its narrowest input because the add-min takes the minimum value of its inputs. This architecture uses a tree structure combined with a list in C++. This is done by using a list of pointers for each node of the tree. In this manner, the tree is accessible in two ways. The first way allows classic tree traversal while the second way uses a second set of pointers as a linked list; thus, an algorithm can go through the whole tree by following pointers of this list. Sometimes it is better to access the tree with the pointer list. Considering the needs of the different processing steps, sometime it is better to access it with the tree pointers. For each component or tap, we associate an ID. This ID is used to link the zones together and will be used to merge decoders.

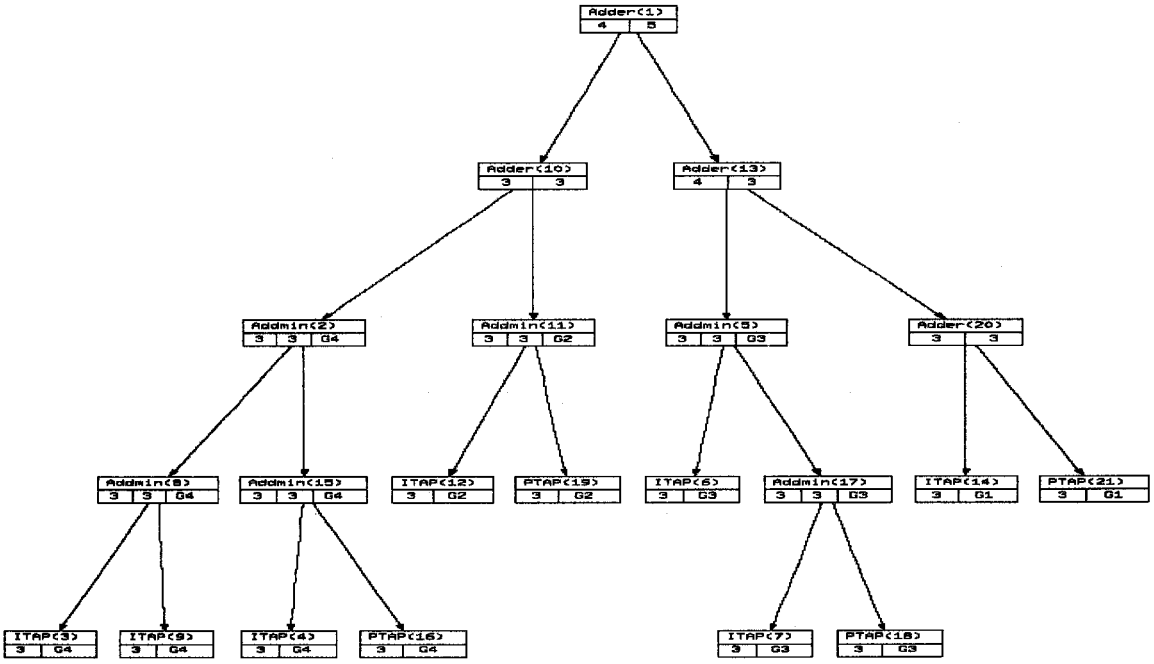

Figure 4.2 The first iteration of a J=4 decoder described as a tree

The list is necessary to search, pass through and merge the trees. The bus width calculation of the bits is done by a recursive function. For an adder, the number of output bits is defined according to the number of bits of its two inputs in order to avoid any overflow at its output. Thus, the output is one bit wider then the widest input. For example, two inputs of 3 and 6-bits wide will force a 7 bit output. The main advantage of this method is the flexibility to merge trees. We will return to this flexibility and to the meaning of ID in Section 4.2.5. Remember that our objective is to merge decoders.

Finally, note that zone 4 is trivially merged, because the SRs are merged directly by making them configurable and the rest of the logic is identical for all decoders.

4.2.4.4 Global representation

As we have seen, the decoder representation is divided into different data structures: the lists that represent the SRs and the tree that describes the arithmetic and logic structure. Each element of the lists and each node of the tree structure have a unique ID and the tap with a given ID in the tree is the same tap as the one with this given ID in either of the lists. This gives the possibility to devise algorithms for optimizing the resources but this is beyond the scope of this paper. Now that the data structures are defined, in Section 4.2.5 we present how to merge decoders.

4.2.5 Decoder Merging

This section shows how to merge decoders in order to implement configurable decoders using the representation described earlier. Merging is done following the two main parts: the logical structure and the heavily pipelined structures. To illustrate the concept, we merge two decoders: J=3 with code 0, 2, 5 and J=4 with code 0, 1, 4, 6. The first step is to do a representation for each decoder as described in Section 4.2.4. The second and final step, the actual merging, uses matrices. We define four matrices: fustemp, fusmap, fusreg, and fusregmap. The first two matrices are used to merge the logical structures and the others are for SR or heavily pipelined structures.

Section 4.2.5.1 describes how to merge the logical structure of all decoders together. Section 4.2.5.2 shows how to merge the heavily pipelined part of all decoders together and discusses the cost to produce a heavily pipelined configurable decoder.

4.2.5.1 Logical structure

The logical structure is the zone 3 illustrated in Figure 4.1 for one iteration of one decoder. We want to merge decoders containing one logical structure each. The first step is to create matrix fustemp from the tree of each decoder. For our example (which merges a J=3 code with a J=4 code), this matrix is as follows (note that this figure is conceptually a single pair of columns but is was drawn using two pairs in order to save space):

Table 4.4 Fustemp matrix ID

J=3	J=4	J=3	J=4
1	**1**	12	12
2	2	13	13
3	3	-1	14
4	4	-1	15
5	5	-1	16
6	6	-1	17
7	7	-1	18
8	8	-1	19
9	9	-1	20
10	10	-1	21
11	11		

In Table 4.4, each of the rows includes one node ID for each decoder. Each column is the information of one decoder. The information in the actual data structure is a pointer to the node but we use the ID here to make it easier to understand how the concept works. For instance, two columns represent two decoders (J=3 and J=4 in our case) and the line containing **1** and **1** (in bold in the table) represents two components for two decoders. The first one represents the pointer of one component of J=3 with the ID=1. The second one represents the pointer of the component J=4 with the ID =1. We arbitrarily chose the last column to represent the code with the greater J value. Also, a value of -1 means that there

is no component because the decoder has fewer components than some of the other decoders.

The second step is to map the address of each node for each decoder in this matrix. The second matrix, fusmap shown as Table 4.5, is used to do this mapping. The content of this matrix is the same as for fustemp but rearranged according to the mapping. Consequently, this representation is used to create the final logical structure of the configurable decoder. For our example, this matrix content is as follows:

Table 4.5 Fusmap matrix ID

J=3	J=4	J=3	J=4
1	1	6	12
-1	2	12	13
-1	3	8	14
-1	4	-1	15
2	5	-1	16
3	6	9	17
4	7	10	18
-1	8	11	19
-1	9	-1	20
7	10	13	21
5	11		

The merging algorithm is simple: the adder trees that are at the root of each tree are mapped together except for the nodes that are not common to all the different decoders, which are mapped according to the trees in which they are present. On the other hand, the add-min modules are assembled in groups that all have a different number of elements [6] (i.e. there is only one group of 2 add-mins, one group of 3 add-mins, etc.) Thus, the groups with the same number of add-mins are trivially mapped together. Likewise, the ITAP and PTAP elements are mapped according to the way their parent add-mins or

adder component were mapped. For instance, the adder with ID equal to 1 is mapped for each decoder as one resource as described by the first row in the fusmap matrix. Each row represents a configurable component or the same component to achieve the given decoder. Again, we use the language **dot** to validate the concept. As shown in Figure 4.3, when a node has two rows, then it is a component that will we used in the two decoders.

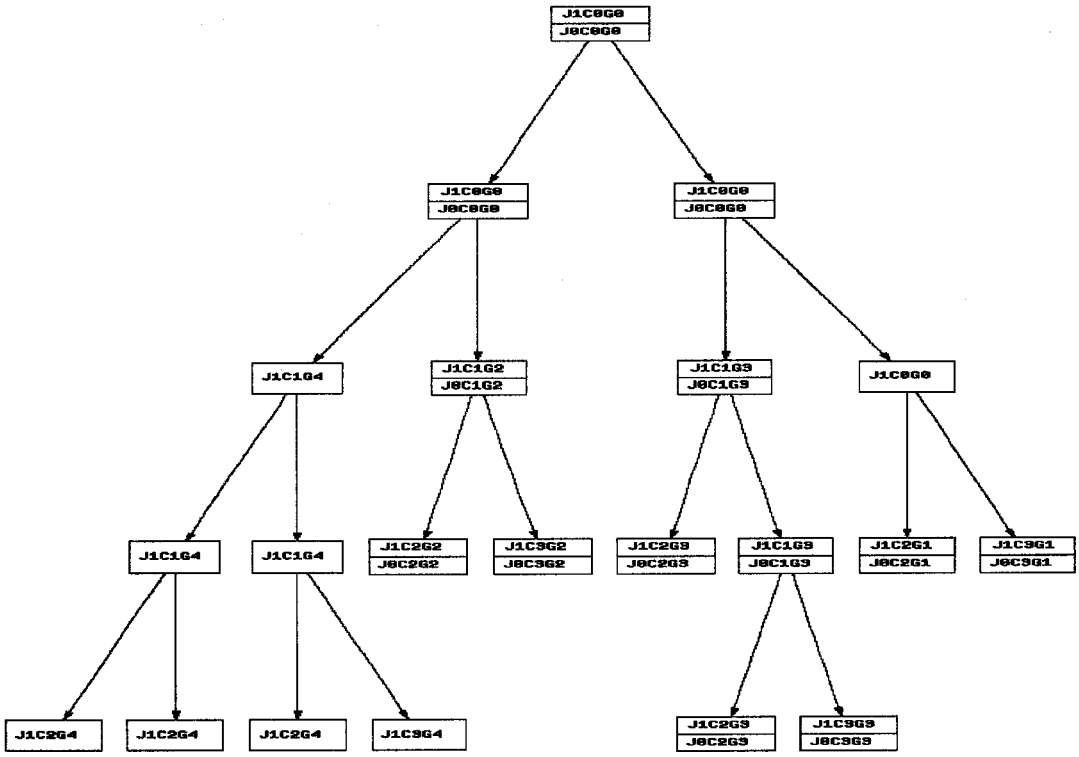

Figure 4.3 Logical structure merge (J=3 and J=4)

The information given in each node of the tree is the decoder identification (J0 or J1), the component type, and the group of each node (G0 to G4). The component types C0, C1, C2 and C3 represent respectively an adder, an add-min, an ITAP and a PTAP.

4.2.5.2 Heavily pipelined structures

This section shows how to merge each heavily pipelined structure (zone 1 and zone 2 of Figure 4.1). Each decoder has more than one SR and we have to merge a lot of those registers with different sizes. The first step of the merging consists of setting a reference representation for each shift register of each decoder. This is done using matrix fusreg, which serves as a cross reference for each decoder. It has one column per decoder and each row describes a SR. It contains the following data items that allow our tool to easily access the following information about each SR.

- Pointer to the input TAP
- Pointer to the output TAP
- Length
- Width
- Flag that states if the SR has been mapped
- Shift register type (ITAP or PTAP)

The second step is to create a second matrix fusregmap to merge SRs together. The relation between fusreg and fusregmap is similar to the one between fustemp and fusmap: the latter is a permutation of the former.

We can merge shift register in many ways. We have explored four of them according to two criteria. The first criterion is whether to merge the SRs that are identical and then use configurable shift registers for the others or to merge using only configurable shift registers. The second criterion is to merge Zone 1 SRs together and Zone 2 SRs together

or merge all Zone 1 and Zone 2 shift registers together. Those methods give similar results because there is a compromise between the complexity of the SR (configurable or not) and of the surrounding logic. Also, the target technology has a strong influence on the compromises thus selecting which method to use should be done on a case-by-case basis

The cost of configurable shift registers is defined in [11]. We merge a SR with another one according to the compatibility of the shift registers. With the pointers in the fusregmap matrix, we can estimate the input and the output cost. Moreover, when we add some technology estimator, we can estimate the configurable hardware cost of each of them. The SRs compatibility assessment is done according to the length and width of the shift registers. In this example, the configurable shift register that we use is the memory based shift register [11]. It allows using the appropriate configurable shift register for a given configuration. For example, a 3-bit wide 1698-stage register can be merged with a 3-bit wide 912-stage register but not with a 3-bit wide 1-stage register. This is due to the shift register constraint described in [11] which imposes that 1698 and 912 stages SRs are realized using a memory and they have at least 4 stages because of the converters involved. Consequently, the structure and the mergeability depend on the technology. Our tool allows abstracting the technology through a C++ virtual class that calls for the given technology that gives the compatibility. With this rule, each element is represented by its ID. The ID is the information that is moved to have the best match possible between the decoders in order to reduce the hardware used. As previously explained, each decoder has two lists that represent heavily pipelined structures. Those lists have an

ID for each element. This matrix element, fusregmap in, contains the information about the shift register source. Remember that a pipeline can be separated in several shift registers. Each row represents a configurable shift register. The input cost is defined by the number of the inputs that are fed to the shift register. The last cost to consider is the output of each shift register. As for the input matrix element, the output matrix element represents the output of each SR but also represents the input of the logical structure. Each input is an ITAP or PTAP as explained earlier.

This section has shown how the representation of more than one decoder can be done and the flexibility to merge them to produce a configurable decoder. As explained, a decoder is created from a specific code. For a given J, we can use different codes. The next section shows how we can choose the suitable code to merge with another one.

4.2.6 Configurable Decoder Exploration

There are many codes with a given value of the J parameter. These codes generate different architectures and have slightly different error correcting capabilities. The methods that we have presented allow producing configurable decoders. This section shows some results from our tool and how we do exploration to find the best set of decoders. The technology abstraction level allows the exploration of the impact of different codes for a chosen technology. Some metrics depend on the technology. In this case, a C++ virtual class is used to give the metrics for a given technology. For instance, the amount of hardware resources at lower level abstraction such as CLB in FPGA is technology dependent. At the opposite, the number of inputs to a SR does not depend on

technology. An estimator is useful to evaluate the resources and the power dissipation. The implemented estimators give a result in minutes rather than in hours (the time needed to synthesize each decoder). Design exploration requires such methods in order to estimate the architecture without spending hours for each decoder.

4.2.6.1 Estimators

Estimators are useful for producing metrics such as the resources and power dissipation. As the architecture is mainly composed of SRs, it is important to estimate the power dissipation and resources of each shift register. Moreover, the logical structure gives the amount of resources that are needed. The resources estimate of each SR is described in [10]. For example, to estimate the resources for VIRTEX-E family, we created a class with the characteristics of resources and power dissipation of this type of FPGA. The estimation of the number of CLB for shift registers is done using two bounds. A lower bound is used to compute the number of LUT, SRL for each SR and to compute the number of CLB. For example, 20 SRL for 5 shift registers is 5 CLBs because each CLB VIRTEX-E has 4 SRL16 thus 20/4 = 5. The second bound is used to compute the number of CLB for each SR. For example, 1.5 CLB for each SR will give 7.5 CLBs for five shift registers.

The estimators are used to merge decoders. Some estimators can be used to quantify the configurable resources that are required to design a configurable decoder. Those estimators allow to compare the merging algorithm used (described in Section 4.2.5.2) and the impact of the resource. Section 4.2.6.2 gives some examples of the capability of

the tool and describes some of the estimators.

4.2.6.2 Example

As an example, this section compares different decoders and the merging cost of different sets of decoders. In this case, the merging criteria used are: ITAPs and PTAPs are not merged together and the merging is done by implementing only configurable SRs (ref. Section 4.2.5.2). The first step is to choose the code and the value of J for the design (J=10 in this example). Our tool supports this decision process by giving power consumption and hardware complexity estimates for different codes; an example of the graph built from the output of our tool is given in Figure 4.4.

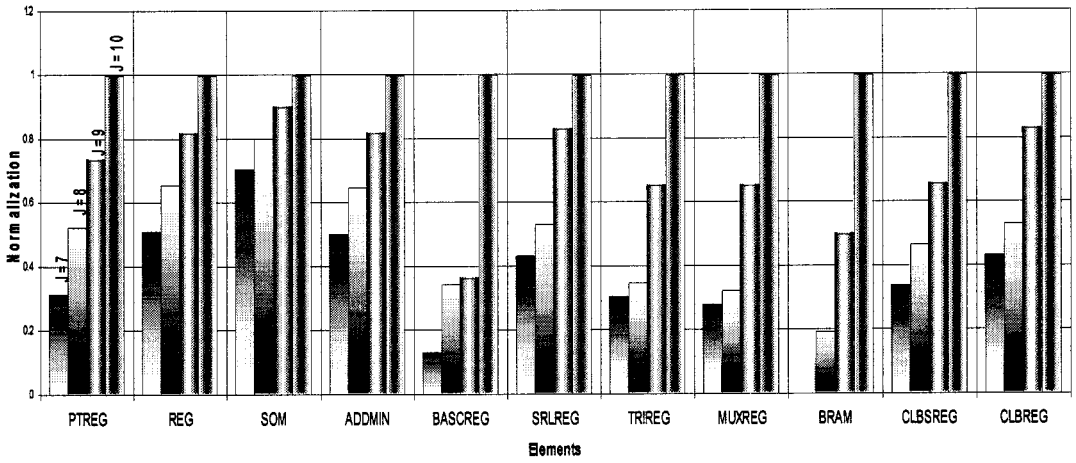

Figure 4.4 Comparison of several decoders parameters for $7 \leq J \leq 10$

As shown in Figure 4.4, several metrics are used to evaluate the resource and power dissipation of the SR. These results tell the designer the impact on the size of the decoder for a given context. This step is thus essential in helping the designer choose the best decoder for delivering the target error performance. The second step is to choose a set of

decoders that will be merged. To this end, our tool will grade the codes of a list given in input. Figure 4.5 shows an example of merging more than two decoders and it shows both cost and cost savings. Six global metrics for 4 different merging combinations of codes with J=6 to J=10 are used.

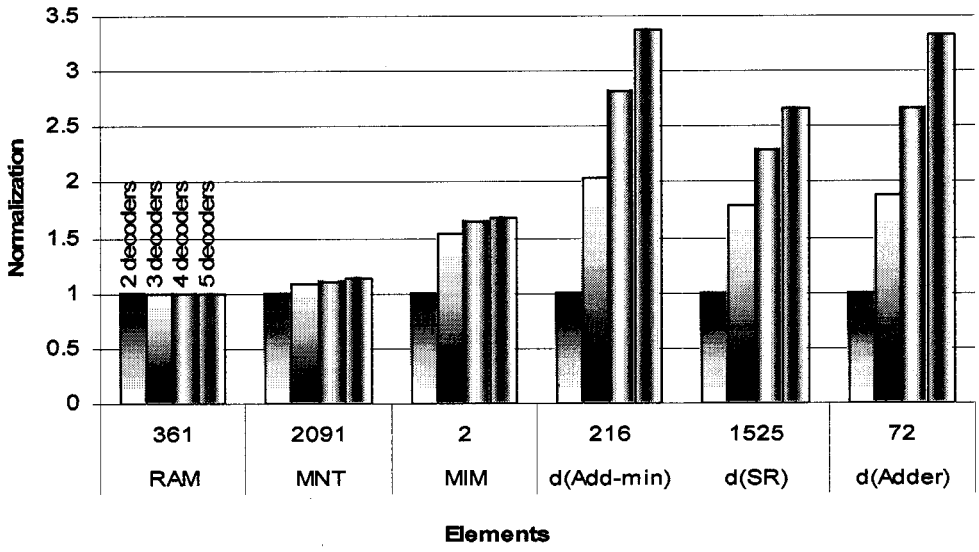

Figure 4.5 2 to 5 decoders merged with 8 iterations

This figure is a normalized histogram where RAM, NMT and MIM show respectively the number of RAMs, 2-input multiplexers and the input ratio on each multiplexer to design a configurable decoder. Also, d(Add-min) d(SR) and d(Adder) represent the resource economy achieved when merging decoders instead of generating them separately. For example, the number 72 represents the value for 2 decoders merging in the d(Adder) category. This means that we save 72 adders by merging 2 decoders. Also in this category, the 5-decoders merging has the greatest economy with a factor of 3.33 (i.e. this merging has 3.33 times the economy of the 2 decoders merging). This method allows comparing any combinations merging with any another. As shown, in Figure 4.5, the

greater cost is for the first 2 decoders while, for the others, the economy is greater and the configurable resource is smaller. It is remarkable that it produces a decoder or a configurable decoder in a few minutes. Also, a configurable decoder allows to have more than one decoder at the lowest cost possible.

Moreover, the SRs, add-mins and adders that we need are reused from a decoder to the other. In general, the configurable decoder will use the resource of the longer code (some of which will have grown in order to be configurable) and the smaller code will reuse elements of the bigger architecture.

4.2.7 Conclusion

In this paper, we have investigated methods to select and implement configurable decoders. This sort of decoder improves the decoding process in a variant channel environment and for research applications. A configurable decoder is constructed by heavily pipelined structures as the available configuration methods are not very suitable in this context. Consequently, we have presented some methods to deal with this kind of structures. A flexible representation of a set of decoders was presented. It uses lists, a tree, matrices and groups. Those groups permit to reduce considerably the complexity of the merging algorithm. We have shown that we can use estimators to give the designer the information about the realization of a decoder and a configurable decoder without the long process of synthesis. New metrics were presented to compare the fusion of a set of decoders. These methodologies provide a new approach to select and design configurable decoders.

CHAPITRE 5

CONCLUSION ET TRAVAUX FUTURS

5.1 Conclusion sur le projet de recherche

Dans cette recherche, nous avons montré qu'il était possible de produire des décodeurs configurables. La première étape nous a permis de créer des registres à décalage à faible consommation tout en étant configurables. Pour ce faire, nous avons investigué les façons avec lesquelles dont nous pouvions les réaliser. Deux nouvelles méthodes ont été proposées. Ces principes ajoutent un peu de matériel tout en réduisant significativement la dissipation de puissance. Les méthodes proposées sont génériques et caractérisées. Par conséquent, une méthode de recherche exhaustive qui permet de déterminer la meilleure structure à utiliser dans un contexte donné a été décrite.

Plusieurs méthodes de conception ont été décrites. Cependant, aucune méthode actuelle ne permet de créer ce type d'architecture dans le cadre d'un flot de conception standard. Nous avons élaboré des méthodes pour le partitionnement de l'algorithme de façon à réduire la complexité de ses implémentations matérielles. Nous avons utilisé le concept de groupe. De plus, à partir d'une représentation existante du décodeur, nous l'avons segmenté en zones et établi des façons pour fusionner les décodeurs. L'utilisation des zones et des ID a grandement simplifié les différentes étapes de fusion. L'architecture configurable créée à partir du plus gros décodeur d'un ensemble considéré augmente l'efficacité d'utilisation des ressources. Avec toutes les nouvelles méthodes présentées, nous avons montré comment sélectionner et générer un décodeur configurable. Ce type

de décodeur augmente la capacité de décodage pour des environnements changeants et les besoins en recherche. Ainsi, nous démontrons qu'avec l'exploration du design et le haut niveau abstraction, nous arrivons à réaliser des décodeurs configurables.

5.2 Limitations et recherches futures

Dans le cadre de ce projet, nous avons jeté les bases sur la réalisation d'un décodeur configurable. L'objectif était la réduction de la puissance et la possibilité de faire un décodeur configurable. Nous avons réalisé ces deux objectifs. Pour y arriver, nous avons réalisé un outil de synthèse pour aider à implémenter des décodeurs configurables ou non. Ceci permet de réduire la dissipation de puissance et d'optimiser l'utilisation des ressources. Nous pourrions avoir comme objectif d'améliorer la vitesse d'opération du décodeur. L'outil que nous avons créé supporte d'éventuels algorithmes tels que les algorithmes d'ordonnancement indiqués dans la revue de littérature. De plus, l'outil fait abstraction de la technologie en utilisant des classes virtuelles. Nous avons réalisé une première version avec le VIRTEX-E qui possède sa propre classe. Éventuellement, il serait important de réaliser d'autres classes pour d'autres technologies. L'outil peut en tenir compte automatiquement. Enfin, nous avons montré la possibilité de faire de l'exploration algorithmique et architecturale. Dans une version future, nous pourrions être intéressés par d'autres estimateurs tels que la vitesse d'opération du décodeur en fonction du code. L'outil est fait de façon modulaire et est très flexible, de sorte que nous pouvons donc imaginer l'ajout de certaines fonctionnalités telles que l'automatisation du choix des décodeurs à fusionner.

DISCUSSION GÉNÉRALE

Dans le cadre de ce mémoire, nous avons jeté les bases pour la réalisation de décodeurs à seuil itératifs configurables. Le décodage à seuil itératif est une approche simple permettant de contourner les problèmes de latence et de complexité du décodage Turbo. Cette réalisation permet d'avoir plusieurs décodeurs dans un design en ajustant le nombre de connexions. Cet ajustement permet de changer de performance dans un contexte donné sans avoir à synthétiser un nouveau design. Dans un premier temps, nous nous sommes concentrés sur l'aspect de la consommation de puissance et de la configurabilité des registres à décalage. Cet aspect était primordial à la réalisation de décodeurs configurables. La consommation de puissance a pu être réduite considérablement en créant deux nouvelles structures : la structure d'activation et la structure avec mémoires et convertisseurs. De plus, une méthode systématique proposée dans ce mémoire permet de choisir la bonne structure pour une configuration de registre à décalage donnée. Ces deux nouvelles façons de faire permettent de réduire la consommation de puissance tout en étant configurable. La configurabilité et la réduction de puissance des registres à décalage permettent d'accroître les possibilités de design de décodeurs configurables et la réalisation d'un décodeur configurable devient possible. Cette première partie a permis d'écrire un article qui correspond au chapitre 3. Dans un deuxième temps, comme les méthodes actuelles ne sont pas adéquates pour cette conception, nous nous sommes concentrés sur les méthodes et sur la modélisation de plusieurs décodeurs afin de les fusionner. De plus, il existe plus d'un code pour chaque décodeur, le concept d'exploration de la fusion afin de trouver le meilleur ensemble de codes pour une

performance donnée a été montré. Cette deuxième partie a permis d'écrire un article qui correspond au chapitre 4. Finalement, nous avons conclu sur les résultats obtenus et les possibilités de cette recherche.

RÉFÉRENCES

[1] ALFKE P., *Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators*, Xapp052 (Version 1.1) July 7, 1996

[2] AT&T, http://*www.research.att.com/~erg/graphviz/info/lang.html*

[3] BECKER J. et al.: *Architecture and Application of a Dynamically Reconfigurable Hardware Array for Future Mobile Communication Systems*, Proc. FCCM'00, Napa, CA, USA, April 17-19, 2000.

[4] BONDALAPATI K. and PRASANNA V. K., *Reconfigurable Computing Systems*, Proceedings of the IEEE, Volume: 90 No 7, July. 2002

[5] BRYANT R. E., CHENG K.-T., KAHNG A. B., KEUTZER K., MALY W., NEWTON R., PILEGGI L., RABAEY J. M. and SANGIOVANNI-VINCENTELLI A., *Limitations and Challenges of Computer-Aided Design Technology for CMOS VLSI*, Proceedings of the IEEE, Volume: 89 No 3, March. 2001

[6] CARDINAL C., *Décodage à Seuil Itératif sans Entrelacement des Codes Convolutionnels Doublement Orthogonaux*, Doctoral thesis, Department of Electrical and Computer Engineering, École Polytechnique de Montréal. June 2001

[7] CARDINAL C., HACCOUN D., and GAGNON F., *Iterative threshold decoding without interleaving for convolutional self-doubly orthogonal codes*, IEEE Transactions on Communications, Vol. 51, No. 8, Aug. 2003, pp 1274 –1282

[8] CHANDRAKASAN A., *Basics of Low Power Circuit and Logic Design, Massachusetts Institute of Technology*, 1997, Pages(s): 58-59

[9] CHEN D. and RABAEY J., *PADDI: Programmable arithmetic devices for digital signal processing*, VLSI Signal Processing IV, IEEE Press 1990.

[10] DUBOIS M., SAVARIA Y. and HACCOUN D., *On Low Power Shift Register Hardware Realizations for Convolutional Encoders and Decoders*, Second Northeast Workshop on Circuits and Systems (NEWCAS) 2004, Montreal, Canada, June 20-23, 2004

[11] DUBOIS M., SAVARIA Y., HACCOUN D., and BÉLANGER N., *On Low Power Configurable and Generic Shift Register Hardware Realizations for Convolutional*, submitted for IEE Proceedings on Circuits Devices and Systems, August 2004

[12] EBELING C. et al.: „RaPiD: Reconfigurable Pipelined Datapath", in [15]

[13] GEORGE M. and ALFKE P., *Linear Feedback Shift Registers in Virtex Devices*, XAPP210 (v1.2) January 9, 2001

[14] GIGLIOTTI P., The Quarterly Journal for Xilinx Programmable Logic Users Xcell 31 - First Quarter, 1999, Pages(s): 9 - 10

[15] GLESNER M., HATENSTEIN R. (Editors), Proc. FPL'96, Darmstadt, Germany, Sept. 23-25, 1996, LNCS 1142, Springer Verlag 1996

[16] HACCOUN D., CARDINAL C., and GAGNON F., *Search and Determination of Convolutional Self-Doubly Orthogonal Codes For Iterative Threshold Decoding*, accepté pour publication dans IEEE Transactions on Communications

[17] HARTENSTEIN R., *A Decade of Reconfigurable Computing: a Visionary Retrospective*, embedded tutorial, Proceedings of the conference on Design, automation and test in Europe, 2001, Pages(s): 642 - 649

[18] LOWY M., *Parallel implementation of linear feedback shift registers for low power applications*, IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, Vol. 43, Issue 6 , June 1996 Page(s): 458 –466

[19] MARSHALL A. et al., *A Reconfigurable Arithmetic Array for Multimedia Applications*, Proc. ACM/SIGDA FPGA'99, Monterey, Feb. 21-23, 1999

[20] MATHUR A. and SALUJA S., *Improved Merging of Datapath Operators using Information Content and Required Precision Analysis*, Annual ACM IEEE Design Automation Conference, Pages: 462 – 467, 2001

[21] MCFARLAND M. C., Parker A. C. and Camposano R., *The High-Level Synthesis of Digital Systems*, Proceedings of the IEEE, Volume: 78 No 2, Feb. 1990

[22] PENG Z., *High Level Synthesis*, Lecture notes, IDA, The Department of Computer and Information Science, LINKÖPING University, *http://www.ida.liu.se/~petel/SysSyn*

[23] WAINGOLD E. et al., *Baring it all to Software: RAW Machines, IEEE Computer*, September 1997, pp. 86-93.

[24] Xilinx, *http://www.xilinx.com/cgi-bin/powerweb.pl*

[25] Xilinx, *Packaging Thermal Management*, XAPP415 (v1.3) August 21, 2003

[26] Xilinx, Xilinx *Virtex Power Estimate Worksheet Version 1.6*, excel version